# Development of object detection system to minimise tripping and slipping risk

Stuart Port

Thesis submitted for the fulfilment of the requirements for the degree of Masters of research

Victoria University, Australia
Institute for Institute for Health and Sport (IHES)

Date submitted: July 2022

# ABSTRACT

Falls within the elderly community are estimated to have cost the Australian Healthcare system approximately $498.2 million in 2001 and are predicted to rise threefold to $1375 million in 2051. A vast majority of these falls are a result of tripping, slipping, and stumbling hazards. These types of falls are commonly caused by everyday household objects such as extension cords, shoes, children's toys, and rugs. The primary reason for these falls among the elderly is the decline in vision and gait functionality.

The wearable technology industry has seen unprecedented growth in recent years, with a market value of $28 billion USD in 2020, and costs are expected to reach $74 billion USD in the year 2026. The current wearable market mainly consists of wearable smart watches and smart glasses, with a vast majority being used in fitness and healthcare applications. A small majority of wearable devices have been introduced into the age care industry, with smartwatches being used to detect falls and alert carers when an individual has had a fall. These devices have only recently been introduced to the market as they rely on low-power cellular components and, as a result, lack the ability to predict and prevent falls. With advances in the Artificial Intelligence industry and the introduction of the tensor processing unit, there now exists practical solutions to implement Artificial Intelligence onto mobile and internet of things devices. This has allowed real-time object detection to be implemented in real-life applications. An area in which these improvements can be taken advantage of is assisted walking for the vision impaired and robotics and assisted walking devices for the elderly.

Due to the lack of fall prevention devices, this research aims to investigate the feasibility of detecting hazardous objects and performing fall prevention from the waist by using a Convolutional Neural Network that is implemented onto an embedded system. In addition, the Convolutional Neural Network will be trained to detect hazards such as spills on various types of flooring. The results of this study will benefit the impaired vision community, robotics community and the elderly population, as it can lead to the development of assisted walking devices

# Declaration

"I, Stuart Albert Port, declare that the Master of Research thesis entitled Development of object detection system to minimise tripping and slipping risk is no more than 50,000 words in length including quotes and exclusive of tables, figures, appendices, bibliography, references, and footnotes. This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work".

"I have conducted my research in alignment with the Australian Code for the Responsible Conduct of Research and Victoria University's Higher Degree by Research Policy and Procedures."

Signature:  Date: 25/07/2022

# ACKNOWLEDGMENT

It is my genuine pleasure to express my deep sense of thanks and gratitude to my mentors and guides, Associate Professor Daniel TH Lai and Professor Rezaul Begg. Their dedication and strong encouragement through Covid-19 times have pushed me to surpass my goals and personal expectations. Furthermore, their timely advice, meticulous scrutiny, scholarly advice and scientific approach have helped me to a great extent in accomplishing my tasks.

I owe a deep sense of gratitude to my parents, Mr Geoff and Mrs Jeannette Port, for always supporting and giving me encouragement through my research period.

I am incredibly thankful to my sister and brother, Ms Megan Port and Mr Sean Port, for keeping me company throughout this time and helping me get through this. Without yous, none of this would have been possible.

Thank you, everyone, for all your support.

# Table of Contents

# List of Figures

# List of Tables

# List of Planned Publications

This these includes chapters that have been published and submitted to the following journals

## Chapter Four:

Port. S, Lai, DTH. & Begg, R. Detection of liquid spills on tiled surfaces via the use of Convolutional Neural Networks. Published in IEEE International Conference on Wearable and Implantable Body Sensor Networks (Accepted)(BSN'22)

## Papers in preparation

1. Detection of liquid spills on carpeted surfaces via the use of Convolutional Neural Networks
2. Detection of liquid spills on tiled and carpeted surfaces via the use of Convolutional Neural Networks

# Chapter 1

## 1.0 INTRODUCTION

Falls within the elder community can be devastating to an individual and result in a loss of confidence when walking, also known as a post-fall syndrome [1]. The fear of falling can restrict an elderly individual's confidence in performing day-to-day activities, resulting in a decline in their quality of life. The primary reason for these falls among the elderly is the decline in vision and gait functionality [1]. The health directive Victoria [2] states that a notable proportion of falls, estimated to be around 79% occur while an individual is transitioning between a bed or toilet and a walking posture or mobile aid devices, such as a wheelchair or walker. This results in 21% of falls being caused by environmental factors such as spills, uneven flooring, or slippery surfaces. Research has shown that elderly individuals' vision is a predominant factor in falls as the reduced vision leads to them not identifying tripping hazards. Most falls within the elderly community can be mitigated by providing a well-lit, clean, obstruction-free environment. The falls within the elderly community have been considered to cost the Australian health care system $498.2 million annually in 2001 and are predicted to rise threefold to $1375 million in the year 2051 [2].

The research conducted within this thesis aims to build a foundation for a vision-based healthcare device that uses artificial intelligence to help detect and alert users to aid in avoiding slips and trips within the elderly community. The proposed research will consist of collecting training data and training a convolutional neural network to detect various coloured spills, along with its deployment onto various embedded devices that feature a camera for object detection. This research will investigate various types of Convolutional Neural Networks (CNN), along with their architecture, such as MobileNet_v1, Mobilenet_v2, Inception_V3 and VGG-16. The computational cost of these models, along with the performance of each model, will be tested on various embedded systems, including the Google Coral and Raspberry Pi. The research conducted can also be adapted to aid other sectors of the healthcare system, such as the vision-impaired or the robotics community.

## 1.2 RESEARCH OBJECTIVE

The research aims to develop a small, low-cost wearable system capable of detecting hazardous objects that pose a slip or tripping hazard. The system will use a camera sensor and a Google Coral Development board to run a real-time Convolutional Neural Network. The system will be placed within a housing that can be attached to an individual's belt at waist level. The system will alert the user to a hazard and provide a means to navigate around the hazard or to proceed through the hazard safely. The main aim of the research can be summarized by the research questions and research aims below. In addition, the research aims and questions below investigate the limitations and core problems when implementing new technology into potential commercial use.

The issues to be investigated in this research project are as follows:

- Is a pre-structured or custom Convolutional Neural Network capable of detecting hazardous liquids and tripping hazards?

- Is the Google Coral Development board a viable option to perform real-time lightweight, portable object detection?

- Determine the efficiency and viability of detecting hazards with the low-cost embedded system when running real-time object detection.

- Test the effects of motion on the unit performance during simulated dynamic motion.

## 1.3 SIGNIFICANCE OF RESEARCH

The research being conducted in this project is significant as falls within the elderly, and visually impaired communities have not decreased in recent years and are expected to increase as the average life expectancy does [3]. The total estimated healthcare cost for falls within the elderly community is predicted to increase from $498.2 million in 2001 to $1375 million in the year 2051 [2]. This research explores the possibilities of incorporating Artificial Intelligence and low-power embedded systems to aid in preventing and minimising falls within these communities through the use of specialised Tensor Processing Units. The proposed device will investigate the detection of common tripping and slipping hazards. Currently, there are no domestic or commercial devices similar to the proposed devices, and the research conducted in this project will help form the foundation to develop these mobility aid devices.

The proposed research in this study will provide the foundation to help bridge the technical challenges evident when performing real-time object detection on low-power embedded systems.

The research conducted in both of these fields has been limited to primarily theoretical and proof of concept studies as both fields have been hindered by technological hurdles. Due to these technology restraints, a limited number of studies have been published, as highlighted in the previous section. The research being conducted in this project will contribute to this relatively new field of study. The proposed research will take advantage of this unique timing and will allow for the research to be among the first in the field of real-time object detection via Convolutional Neural Networks on a Tensor Processing Unit-based embedded system. The published research could provide other researchers with insight into how effective this new technology will be, and the challenges endured when used in different environments. In addition, the research being conducted within this study can also be applied to other technology fields, such as robotics and healthcare.

## 1.4 OUTLINE OF THESIS

- Chapter 2 – This chapter reviews the literature on the current state of the elderly and vision impaired community within the healthcare system who are prone to accidental falls, along with current technology in the area. The literature also investigates the current hardware and software available to the market and the various types of architectures that are commonly found in standard Convolutional Neural Networks. The various characteristics of convolutional neural networks are surveyed, including the training methods, activation functions and common issues.

- Chapter 3 – This chapter contains the methodology of the steps undertaken to develop an assisted walking device, along with the methodology of training the related machine learning models. This chapter will include the method used for image collection to form the training and testing datasets and the labelling and categorisations of classes. The training methodology used to train and test the convolutional neural networks will be included in this section, along with the methods used to increase the performance of the selected models. The quantisation of models and installation onto the Google Coral Development board will be documented in this chapter.

- Chapter 4 – This chapter presents the methodology, results, and discussion of the experiment of split liquids on tiled and laminated surfaces. This chapter explores the training and testing dataset and the method used to train and evaluate the convolutional neural network architectures. The quantisation of each model will also be conducted, and the results of both model will be discussed. It was discovered that Inception_V3 was the highest performing architecture during training, achieving an accuracy of 96%. However, MobileNet_V1 achieved the highest testing accuracy of 88%. During quantisation, it can be seen that MobileNet_V1 achieved the highest accuracies in both training and testing, resulting in accuracies of 86% and 85%, respectively.

- Chapter 5 – This chapter presents the methodology, results, and discussion of the experiment of split liquids on carpeted surfaces. This chapter explores the training and

testing dataset and the method used to train and evaluate the convolutional neural network architectures. It was found that Inception_V3 performed higher than MobileNet_V1 during the initial training, with a training accuracy of 94%. It was seen that MobileNet_V1 achieved the highest accuracy when applied to the training dataset, with an accuracy of 91%. MobileNet_V1 achieved the highest accuracy during quantisation during training and testing, resulting in 88% and 87%, respectively.

- Chapter 6 – This chapter presents the methodology, results, and discussion of the experiment of split liquids on tiled, laminated, and carpeted surfaces. In addition, this chapter explores the combination of chapters 4 and 5, the training and testing dataset, and the method used to train and evaluate the convolutional neural network architectures. The quantisation of each model will also be conducted, and the results of each model will be discussed. During training, Inception_V3 achieved the highest training accuracy of 93%. However, MobileNet_V2 achieved the highest testing accuracy of 92%. During quantisation, Inception_V3 achieved the highest accuracy of 96% during training. While MobileNet_V1 achieved the highest in training with an accuracy of 89%.

- Chapter 7 – This chapter presents the methodology, results, and discussion of the application of the selected embedded systems in a simulation motion test. In addition, this chapter explores the viability of using a Convolutional Neural Network model on an embedded systems platform during a simulated motion test. It was determined that a Convolutional Neural Network could detect liquid spills operating on an embedded system. Furthermore, it was found during testing that the Google Coral Development Board outperformed the Raspberry Pi on machine learning tasks and power consumption.

# CHAPTER 2: LITERATURE REVIEW

## 2.1 ELDERLY FALLS BACKGROUND

A report published by the Australian Commission on Safety and Equality in Health Care [3], found that between 2008 and 2009, 33% of elderly Australians over the age of 65 experienced a severe fall resulting in injury, with 10% of these falls requiring hospitalization. The vast majority of falls are associated with intrinsic risk factors such as weakened muscles, medication, poor eyesight, and unsteady gait or use of mobility aids. These intrinsic factors make up 79% of falls, and the other 21% are caused by environmental hazards such as spills, rugs, and uneven surfaces. It is stated that a notable proportion of falls within the 79% occur while an individual is transitioning between a bed or toilet and a walking posture or mobile aid devices, such as a wheelchair or walker. This study highlights that 79% of falls that occur from transitioning cannot be helped without the aid of rails or manual assistance from carers. The remaining 21% of falls caused by environmental factors are thought to be the main result of the individual having poor eyesight and poor spatial awareness resulting in not detecting hazardous objects when walking. The report emphasizes that these falls can be easily reduced if the individual is aware of the hazard and manoeuvres around them.

An article written by Health Direct Victoria [2] claims that an average of 1 in 3 elderly Australians will experience a severe fall every 12 months, with 1 in 5 of these falls requiring hospital attention. The article also states that 2 in 3 falls are caused by environmental factors, such as poor lighting conditions, slippery surfaces and spills, uneven surfaces such as steps and tripping hazards such as rugs, extension cords and uneven carpets. This study demonstrates the importance of various environmental factors that can lead to the cause of falls for the elderly population of Australia. The article emphasizes that falls can be vastly reduced by creating a safer environment for the individuals however acknowledges that in most situations, this cannot always be accomplished.

It is currently estimated that 300 million individuals worldwide suffer from debilitating vision impairments. A study published in 2020 by Ava K Bittner et al. [4] investigates the impact on the quality of life that vision impairment individuals suffer from their condition when aid is not provided. The research investigates the limited or lack of available resources to help teach visually impaired individuals mobility skills through a telehealth system. The research

demonstrates that mobility skills for the visually impaired are vital and states that their quality of life drastically suffers from the loss of their mobility. The research also states that mobility aid equipment such as walking canes and floor markers are crucial for the individual, and further assistive devices can be used to increase their mobility and quality of life significantly. This article indirectly identifies the gap between possible potential assistive technology and the improvement of the quality of life for a visually impaired individual.

The vision-impaired community also suffers from similar trips, slips, and falls as the elderly. Unfortunately, there is currently not readily or publicly available published research that provides data on the injury rate of the vision-impaired community. In addition, it is believed that fall data for the visually impaired community is not collected as they are recorded under the general populace and cannot be defined by age, unlike the elderly community.

It can be seen that there are programs and research that aim to improve the navigation and mobility of vision-impaired individuals, such as the study published by Gianni Virgili and Gary Rubin [5]. This study investigates and emphasizes the importance of improving the mobility of visually impaired individuals. The research looks at providing the skills required to maintain travel independence and teaches them new orientation and mobility skills to compensate for the reduction of visual stimuli. This research demonstrates a need to improve mobility for visually impaired individuals, and it is believed that it can be significantly increased with wearable devices that provide navigational assistance.

## 2.2 Assistive Technology for Fall Prevention

As it currently stands, a limited amount of assisted walking devices for the visually impaired and elderly take advantage of artificial intelligence. Most equipment used within this field is considered to be manual and does not possess the means to benefit the user with cutting-edge technology. Most equipment within the age care community focus on detecting the fall of an elderly individual and not on the prevention of the fall. Equipment such as headsets, walking canes, and smart insoles have been used to observe bio signals [6] that can indicate an elderly individual is about to experience a fall but are only capable of predicting a fall seconds before it occurs.

One device currently on the market is the OrCam [7], designed to assist visually impaired individuals with detecting household objects and reading texts. The device is a portable camera that attaches to a pair of glasses and uses a Machine Learning Model to analyse a single frame image. The OrCam is a button-operated device that does not include the ability to operate a continuous live video and requires the user to stop for a brief period so the device can analyse its surroundings. Customer reviews state that this device takes a significant amount of time to respond to users and is speculated to be caused by the lack of processing power available on the unit. Due to the unit's size, the device designers would have had to reduce processing power to conserve the operational time of the device. Issues such as the lack of processing power and battery life are common on portable devices. The current iteration of the device would not be sufficient in the application of real time object detection. With the implementation of a Tensor processing unit, a portable device will be capable of analysing a constant video feed. Being able to analyse a constant video feed will enable real-time object detection and can alert the user to potential slipping hazards. This approach leads to many benefits over the OrCam as the user is not required to stop and activate the camera to analyse a single frame.

## 2.3 HARDWARE

### 2.3.1 EMBEDDED SYSTEM PLATFORMS

There are currently multiple major companies competing in the edge hardware market, with Google, Nvidia and Intel leading the market. Edge computing devices specialise in running Artificial Intelligence models on low-power systems such as the Google Coral [8], Nvidia Jetson [9] and NCS [10]. Google has typically been leading the market with the deployment of the Tensor Processing Unit on their Google Coral Development Board and Google Coral USB Accelerator. The Nvidia Jetson and NCS also feature their own machine learning accelerators with Graphical Processing units and Vision Processing Units, respectively. The Google Coral Development Board and Nvidia Jetson share the same form factor as the Raspberry Pi, while the Google Coral USB Accelerator and NCS share the same form factor as a standard USB. A machine learning inferencing benchmark [11] can be seen below in Table 1, where the inferencing speed (milliseconds) is the time taken to process and analyse an image. It should be noted that the lower the inferencing speed, the better performance is provided.

| Board | MobileNet v1 (ms) | MobileNet v2 (ms) |
|---|---|---|
| **Coral Dev Board** | 15.7 | 20.9 |
| **Coral USB Accelerator** | 49.3 | 58.1 |
| **NVIDIA Jetson Nano (TF)** | 276.0 | 309.3 |
| **NVIDIA Jetson Nano (TF-TRT)** | 61.6 | 72.3 |
| **Movidius NCS** | 115.7 | 204.5 |
| **Intel NCS2** | 87.2 | 118.6 |
| **MacBook Pro** | 33.0 | 71.0 |
| **Raspberry Pi** | 480.3 | 654.0 |

**TABLE 1: BENCHMARK COMPARISON OF INFERENCING SPEEDS WHEN COMPARING EDGE COMPUTING DEVICES.**

The power consumption of the devices also plays an essential role in making the device mobile. For example, it can be seen in Table 2 [11] below that the Google Coral Development Board has a relatively lower power consumption compared to other devices. Therefore, the

Google Coral Development Board provides a low power consumption when considering the inferencing speed, making it the optimal choice for the proposed research.

| Board | Idle Current (mA) | Peak Current (mA) |
|---|---|---|
| **Coral Dev Board** | 600 | 960 |
| **Coral USB Accelerator** | 470 | 880 |
| **NVIDIA Jetson Nano** | 450 | 1220 |
| **Movidius NCS** | 500 | 860 |
| **Intel NCS2** | 480 | 910 |
| **MacBook Pro** | 1570 | 1950 |
| **Raspberry Pi** | 410 | 1050 |

**TABLE 2: BENCHMARK COMPARISON OF CURRENT CONSUMPTION WHEN COMPARING EDGE COMPUTING DEVICES.**

Various sensors such as infrared and ultrasonic range finders, Lidar and imaging sensors are capable of detecting the hazardous objects required within the scope of the research. Infrared and ultrasonic range finders and Lidar only provide the distance to an object, with Lidar providing a two-dimensional point grid with multiple distances to an object. These sensors yield strong positive results when a solid object is placed in their path. However, due to the nature of this research, these sensors will not be capable of detecting liquids. Liquid spills are considered to be challenging to detect from a distance as they are typically transparent or reflective. We believe that the recommended course of action is to detect the liquid with imaging sensors that will provide data on the shape and colour of the liquid. Sensors such as infrared range finders will only provide the distance to the liquid, yielding the same result as the surfaces around the liquid.

### 2.3.2 GOOGLE CORAL DEVELOPMENT BOARD

The Google Coral Development Board is Google's entry into the single-board computer edge market [8]. The Google Coral Development Board takes a similar form factor to the Raspberry Pi and NVidia Jetson. The board is designed to perform fast machine learning inferencing and specialises in the ability to scale a prototype to production. The development board features an on-board Edge TPU coprocessor, which is capable of performing 4 trillion operations per second, with every trillion operations using 0.5 watts. The addition of this chip drastically reduces the time a Neural Network requires to process data. It is reported that the board can run models such as MobileNet_V2 at 400 frames a second, making real-time object detection possible. The board features NXP i.MX 8M SoC, which consists of a quad Cortex-A53 and a Cortex-M4F, with LPDDR4 RAM. The board also has on-board wireless and Bluetooth 4.2, along with general purpose in/out pins (GPIO). The device is powered by a 5-volt 3-amp DC power supply. The Google Coral Development Board is a Linux-based system that features Google's custom Mendel operating system, which is designed for use with TensorFlow Lite Neural Networks. This increased processing power, along with the reduction in energy consumption, allows the device to process large amounts of data on board without transferring data over the internet or to another more powerful device.

### 2.3.3 GOOGLE CORAL USB ACCELERATOR

The Google Coral USB Accelerator is a USB-C device that provides an Edge TPU as a coprocessor for the host system, while enabling high-speed machine learning inferencing. The USB Accelerator's design is mirrored off the Google Coral Development board allows for the incorporation of the tensor processing unit to any system that operates Linux, Mac and Windows. This allows for single-board computers such as the Raspberry Pi to have drastically increased processing power when faced with machine learning challenges. The system is designed to be used as a plug in and play solution, where users are only required to install the Edge TPU runtime repository. The Accelerator shares the same performance as the Google Coral Development Board, enabling 4 trillion operations per second while consuming 0.5 watts per 1 trillion operations. It is stated that the Accelerator typically runs at 20 to 40 times faster than an embedded systems CPU [12].

### 2.3.4 RASPBERRY PI 4B

The Raspberry Pi 4 model B shares many similarities with the Google Coral Development Board and the NVidia Jetson. The Raspberry Pi 4 Model B is the first of the fourth generation of Raspberry Pi computers, which offers an increase in both computing and memory speeds over the previous models. The Raspberry Pi 4B is a well-known low-cost, Linux-based computer aimed at hobbyists and entry-level users in the microprocessors market. The Raspberry Pi 4B hosts the ARM Cortex A72 microprocessor which runs at a speed of 1.5GHz [13]. This CPU runs slightly faster than the previous generation of the Raspberry Pi's but delivers approximately three times the performance. The system also features LPDDR4 Ram and is considered comparable to entry-level desktop computers. The system features a Broadcom VideoCore VI as its graphical processing unit. This processing unit isn't designed for machine learning tasks and as a result the performance of the Raspberry Pi on machine learning task is poor. The Raspberry Pi 4B is considered to be a power efficient device with a maximum power supply of 5.2 volts at 3 Amp and an idle power consumption of 3.8 watts and 6.0 watts at full load [14].

### 2.3.5 NVIDIA JETSON NANO DEVELOPER KIT

The Nvidia Jetson Nano Developer Kit is a single-board computer similar to the Raspberry Pi and Google Coral Development Board. The Jetson Nano is similar to the Google Coral Development board and is designed for ease of use for beginners and developers looking to run high inferencing machine learning models. The Jetson Nano shared a similar formfactor to the Google Coral development board and features a production ready system on module that performs at 472 gigaflops per second at 5 watts a cycle [9]. The system on module chip consists of a Quad core Cortex-A57 processor and a 128 CUDA Cores AI Accelerator based on the Maxwell architecture. CUDA Cores are typically found within Nvidia GeForce Graphical Processing Units and share similar architecture to a CPU core and specialise in fast and efficient parallel computing. The Jetson Nano allows for the use of both TensorFlow Lite and Pytorch models, whereas the Google Coral development board only allows for TensorFlow Lite models. The board also supports Nvidia TensorRT accelerator library which allows for models to use 16-bit floating points for the weights and biases for their models, whereas the Google Coral Development board only allows for 8-bit integer quantised models. Due to the flexibility of the Jetson Nano, it is possible to achieve higher accuracies with the board using the same model as the weights and biases can be more detailed [15].

### 2.3.6 Tensor Processing Units

As the Tensor Processing Unit is a relatively new hardware item, there is a limited amount of published journal articles involving the use of the technology. Most publicly available information is provided in the form of grey literature from sources such as hardware manufacturers or companies that have implemented the hardware. The lack of information and research conducted is mainly due to the Tensor Processing unit only being made publicly available in mid-2020. The Tensor Processing Unit allows for the computation of tensor units, a form of a simplified matrix. Traditionally, Central Processing Units allow for the calculation of two scaler units, and a Graphical Processing Unit allows for the calculation of two vector units [16]. A Tensor Processing unit's unique tensor calculations are ideal for meeting the structural requirements of various machine learning models. Prior to the release of the Tensor Processing Unit, researchers investigated the application of Convolutional Neural Networks on traditional embedded systems that feature a Central Processing Unit. The Convolutional Neural Networks that were implemented onto traditional embedded systems experienced issues related to the system's computational power. This has reduced most studies to using Convolutional Neural Networks to analyse a single frame instead of a live video feed. The embedded systems that host a Tensor Processing Unit provide a substantial performance increase compared to the conventual embedded system noted in the aforementioned journal article. In addition to this, the difference in performance is illustrated in the journal article published by N. Jouppi et al. [17]. This article investigates the performance increases that a Tensor Processing Unit contributes to the field of embedded systems. This article can be used to demonstrate a benchmark for the comparison between the two categories of embedded systems. It is evident within the article that a Tensor Processing Unit developed and used by Google yields a performance increase, displaying a speed increase of 15-30 times and being 30-80 times more power efficient. This journal article ultimately demonstrates the vast improvement that a Tensor Processing Unit can provide. This improvement will provide enough computational power to allow the system to analyse a high frame rate live video, which is required for applications such as object detection for assisted walking devices. Once again, however, a limited number of published studies investigate Tensor Processing Units, and it is difficult to obtain an accurate portrayal of the technology.

## 2.4 SOFTWARE

There is currently a wide variety of Artificial Intelligence architectures that can be applied to object detection and object recognition scenarios, with each architecture providing benefits and drawbacks depending on the situation it is used in. The effectiveness of various machine learning models can be seen in the journal article published by D. Shadrin et al. that compares each model's performance in identifying seeds during the germination phase [18]. This work explores the performance of various machine learning models such as Multi-layer Perceptron, Support Vector Machines, Decision Trees and K-nearest Neighbourhood. The research was designed to test the effectiveness of each model by recognizing and distinguishing the difference between the different stages of seed germination. The system used still low-resolution images to detect the different stages of seed germination. The system, however, did not have the required memory or computing power to store multiple images or live videos. As a result, the system was able to detect the different germination stages at high accuracy, displaying that Multi-layer Perceptron is the ideal machine learning model that can be used for this application. The Multi-layer Perceptron is the predecessor of the Convolutional Neural Network [19]. It is demonstrated that Multi-layer Perceptron and Convolutional Neural Networks are the predominated deep learning models required to provide the best results for object detection. This journal article can be used to demonstrate that embedded systems are capable of running advanced machine learning models. However, most embedded systems lack the computational power and memory to run machine learning models.

Convolutional Neural Networks have made recent strides across multiple areas such as computer vision and natural language processing and have been labelled the predominant structure for object detection and object recognition [20]. This deep learning model provides low latency and highly accurate object detection. The structure is comprised of a traditional Neural Network, along with a convoluting layer. The convolutional layer is used to extract features of an image, such as edges and colour variations, along with compressing and averaging the image's pixel data. The extracted features are then fed into a Neural Network consisting of layers and arrays of tuneable nodes. The data provided by the extraction layer is used to activate a series of nodes which in turn activate a final output node providing the label for the detected object. There is currently a variety of pre-made structured Convolutional Neural Network

models, such as the YOLO V2 [21], Mobile net V1 [22] and Resnet [23]. These models are freely availed to be used in transfer learning and specialised in detecting common everyday objects such as cars, chairs, humans and more. However, these models do not specialise in detecting one class of the object and may lack the requirements of detecting the spills and hazardous objects required in the scope of this research. There is also considered a large number of datasets available online such as Kaggle [24] and ImageNet [25], which provide prelabelled images of various object classes. These datasets mainly consist of common everyday objects, such as trains, cars, chairs, and animals. The datasets do not provide any labelled images of liquids or spills and are considered to be marketed at hobbyists and AI enthusiasts and are not meant for commercial use.

The environmental factors that the system will encounter have been moderately explored in various research papers. The main environmental factor that will challenge the system is the lighting conditions of objects perceived from different viewing angles. This is due to shadows and direct lighting changing the pixel values of the object and obscuring the image for the deep learning model. These effects have been explored from a stationary system in various research papers that involve the use of three-dimensional object detection under arbitrary lighting conditions [26]. This paper capitalises on the prospective of visual effect artists specialising in three-dimensional objects. The first study investigates the detection and identification of objects that are perceived from different lighting positions. The study primarily investigated the mathematics and theory behind the encountered lighting problems. This test was conducted from stationary positions and did not involve any moving components. This paper identified the challenges in training a Convolutional Neural Network to perceive objects in various lighting conditions. Environmental factors such as an uncontrollable background also affect how Convolutional Neural Networks perform. These factors can drastically impact the performance and accuracy of a Convolutional Neural Network [27]. These challenges are noted when performing object detection in an uncontrollable environment. This paper focussed on the accuracy and evident challenges when detecting objects in the foreground of the scene while having a complex moving background. The paper yielded positive results and has identified some factors that will aid in distinguishing objects while in a complex moving environment. It can be noted that both of these studies were conducted using conventional computers and not

embedded systems. Therefore, the accuracy level obtained in these studies is not guaranteed to be obtained when using an embedded system.

There are currently no publicly available studies that investigate an object detection system mounted to an individual or assisted walking device that uses a Convolutional Neural Network. The closest studies that can be found are object detection systems being mounted onto autonomous vehicles or drones. The challenges associated with mounted object detection on portable devices are mainly in drones [28]. This study does not directly impact or involve an exoskeleton or assisted walking device. However, it depicts challenges such as the lack of computational power, detecting and identifying moving objects and various environmental factors that can be analysed and adapted to help develop object recognition for an exoskeleton or assisted walking device.

There appears to be no research published that directly involves the research questions 1 and 2 stated in section 2, Research Aims. However, there is a limited to moderate amount of research that investigates some areas that will be beneficial to the project. There is also no grey literature that involves the use of object detection in the navigation of exoskeletons and assisted walking devices. It is strongly believed that this new field of research has been hindered due to the lack of hardware with the required computational power. This research aims to fill this gap by implementing real-time detection of the latest hardware, namely the Google Coral Development board.

### 2.4.1 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN) is a well-recognised deep learning architecture that has allowed for real-time object detection and objects classification. The concept of CNN stemmed from the research published by Hubel & Wiesel [29] in 1959, where it was discovered that cells within the visual cortex of animals are responsible for detecting light from their respective receptive fields. The first digital adaptation of this research was proposed by Kunihiko Fukushima and Sei Miyake in 1980 [30]. Kunihiko Fukushima and Sei Miyake proposed the recognition, which can be regarded as the predecessor of the Convolutional Neural Network. In 1998, this architecture was further adapted by LeCun et al. [31] and developed into a multi-layer artificial neural network labelled LeNet-5. This architecture is considered the foundation of the modern framework of Convolutional Neural Networks. The LeNet-5 was further improved upon by other architectures such as AlexNet [32], VGG-16 [33], GoogleNet and ResNet [34], with each evolution of architectures providing new alternative functions to compute image and video classification.

Since the development of the LeNet-5, the computational power available to train and execute Convolutional Neural Networks has drastically increased. This increase in computational power has allowed Convolutional Neural Networks to perform object detection and object recognition in real-time. These improvements to machine learning architecture and computational power have led to machine vision software being deployed in both commercial and domestic applications. As a result of the demand for machine learning applications, the new field of edge computing has arisen [8], where data is processed at the edge of the typical computer paradigm. Edge computing has encouraged the development of high-performance power-efficient devices, such as products such as the Google Coral and Nvidia Jetson. These devices allow for quantised versions of Convolutional Neural Networks to be run on embedded systems. These improvements in software and hardware make it possible to perform lightweight object detection in real-time on mobile devices.

### 2.4.2 Structure of a Convolutional Neural Network

Convolutional Neural Networks are a subdivision of Neural Networks and are predominantly used in image classification and object detection within images and videos. The fundamental building blocks of a Convolutional Neural Network consist of Convolutional Layers, Pooling Layers, Activation Layers and Fully Connected layers. The model can be broken down into three sections, feature learning, fully connected layers, and binary classification. Figure 1 below shows an example of a typical Convolutional Neural Network, which inputs an image and performs feature extraction, along with the fully connected layers and binary classification.

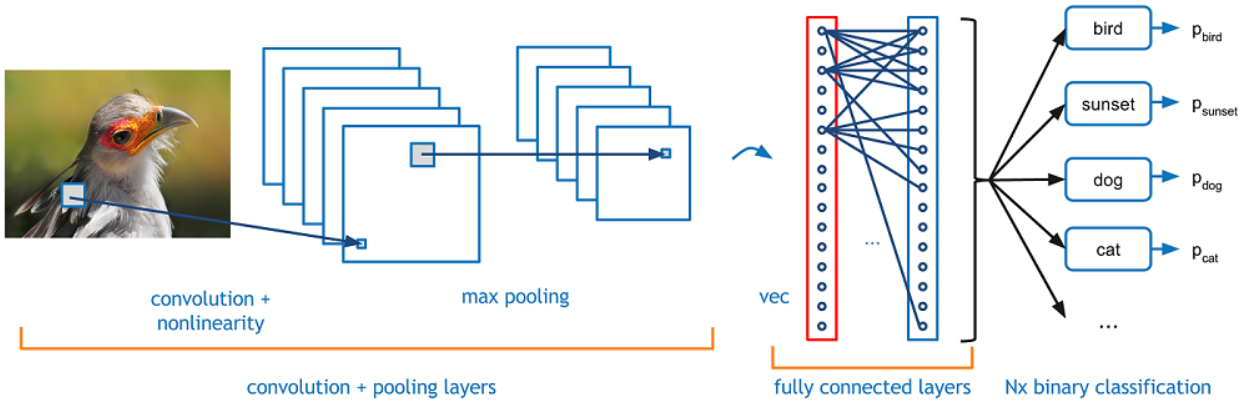



FIGURE 1: BASIC STRUCTURE OF A CONVOLUTIONAL NEURAL NETWORK **[35]**

#### 2.4.2.1 Input Layer

The input layers of Convolutional Neural Networks are typically images or a video stream, as both are essentially an array of pixels. If a video is provided to the convolutional neural network, then the algorithm will select the first frame of the video and analyse it as a still image. Each pixel within the array has a numeric value within the ranges of -255 and +255. This number represents the colour value of each pixel within the array. Coloured images typically consist of 3 colour arrays that represent the red, green, and blue spectrum of the image, with some image formats featuring a fourth array which consists of information regarding the intensity or shade of an image. Once the image has been analysed, then the algorithm will select the following image or frame within the video stream.

2.4.2.2 Convolution layers

Convolutional layers are the core building block of most Convolutional Neural Networks and are responsible for the increased performance in object detection when compared to regular Neural Networks. A Convolution can be considered to be a typical moving filter that is applied to an image that results in obtaining information based on its set of activation parameters. A Convolutional filter is referred to as a kernel and is usually a square grid of 3x3, 5x5 or 7x7 pixels that move across the input image. The kernel then performs a mathematical function such as averages, adds, and multiples or performs logical functions such as comparisons on the values of the pixels within the grid. It is common practice to have multiple kernels run over the input image, as each kernel can be used for various feature extractions. This is done to detect different shapes, such as straight or diagonal edges of objects, or to identify a change in colour between two objects. The outcome of the mathematical operation is then stored in another grid referred to as the convoluted feature map. Since the size of the output feature map depends on the filter size, the number of features extracted per layer can be altered by changing the dimensions of the kernel. The size of the kernel filter usually depends on the amount of detail that is required to be detected from the original input image. The process of extracting features from an image happens through the Convolutional Neural Networks Convolution layers. This process is illustrated in Figure 2.

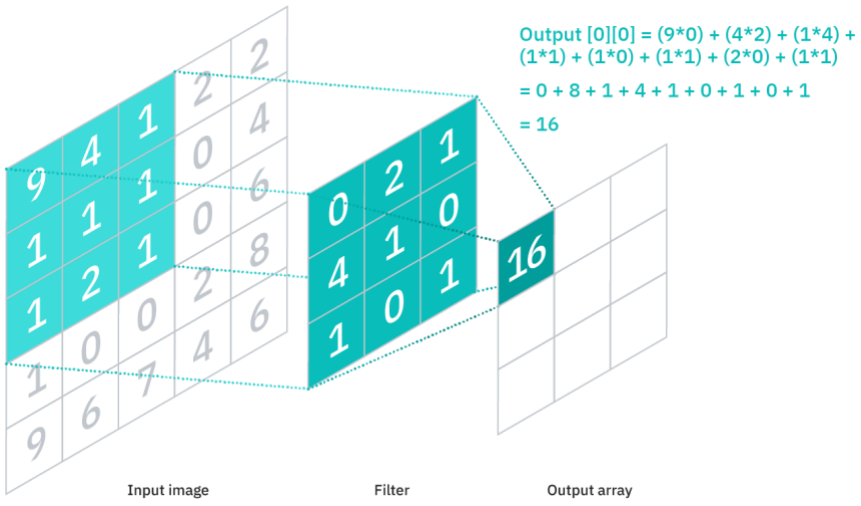



FIGURE 2: STRUCTURE OF KERNEL FILTER **[36]**

2.4.2.3 Pooling layers

The pooling layer is responsible for the reduction of the spatial size of the convoluted feature. Pooling is required to reduce the computational cost of the model and is achieved through summarising and reducing the data through the dimensional decrease of the feature map. Pooling is also included within the model to help with separating predominant feature aspects, which are rotational and positional invariant. Typically, two types of pooling layers are used within a Convolutional Neural Network, referred to as Max pooling and Average pooling. Max pooling is used to return the maximum value from the observed section of the feature map that is covered by a kernel. Max pooling can additionally be used as a method of noise suppressant within the feature map. In comparison, Average pooling restores the average value from the section of the feature map that is covered by the kernel. Average pooling only performs the dimensional decrease and continues to include noise and other impurities throughout the network. Max pooling is the preferred method and is commonly used in most Convolutional Neural Network architectures. The difference between max pooling and average pooling can be seen below in Figure 3.

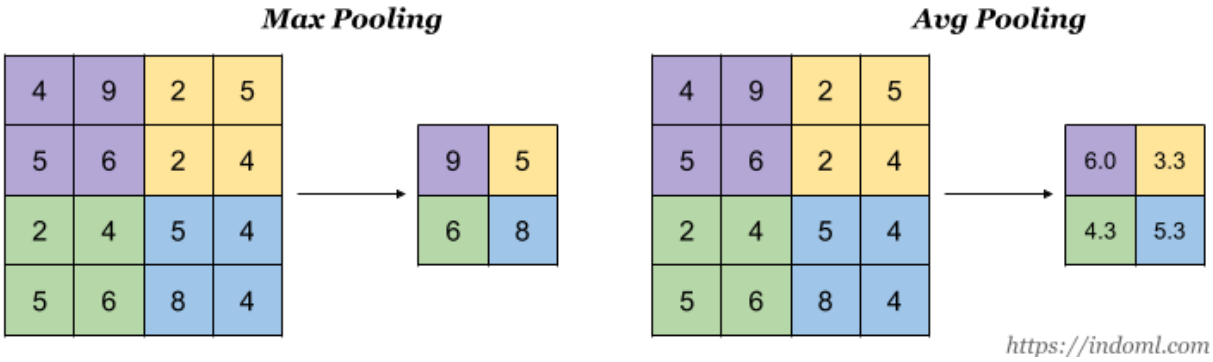



FIGURE 3: EXAMPLE OF MAX POOLING COMPARED TO AVERAGE POOLING **[37]**

The Convolutional layer and the additional pooling layer form the first section of the convolutional neural network. It is common practice to have networks perform multiple convolutional layers and pooling layers in series to simplify feature extraction and reduce computational costs.

#### 2.4.2.4 Fully Connected Layers

Fully connected layers are the foundations of Convolutional Neural Networks and were derived from its predecessor, the Neural Network, which has demonstrated its ability to perform low-end image classification and object detection. The architecture of the Convolutional Neural Network begins with the Convolution and pooling layers, which extract the features from an image and autonomously build a feature map for each input image. The consequence of the Convolution cycle forces the fully connected layer to be introduced in the last stage of the model. The target of the fully connected layer is to take the information gathered within the Convolution and pooling cycle and use the gathered information to identify the object within the image. The fully connected layers also help the model to make sense of the features that were extracted in the convolutional and pooling layers and provide a class label at the binary output stage. For instance, an image of a dog will have features extracted such as ears, tail and legs during the Convolutional and pooling layers, and the fully connected network will use this information to determine if the output label for the object should be a dog or a bird.

At the end of the Convolutional and pooling cycle, the features are removed from the feature map and are flattened into the input layer of the fully connected network. These input values then stream into the main layers of neurons. Each signal from the input layer is increased or decreased depending on the weights it passes through and then continues onto the next weight depending on if the activation function is triggered. The weights and activation functions determine the strength of the signal before it reaches the net set of neurons. The weight of each neuron is altered during the backpropagation cycle during the training of the Convolutional Neural Network, and the value of neurons is typically randomised. Once the signal reaches the final fully connected layer of neurons, it is passed to the binary classification layer, which provides a label to an object in the input image. The model then selects the neurons in the binary classification layer with the highest total sum as the most probable label for an object in the image. The fully connected layer can be seen in Figure 4 below.

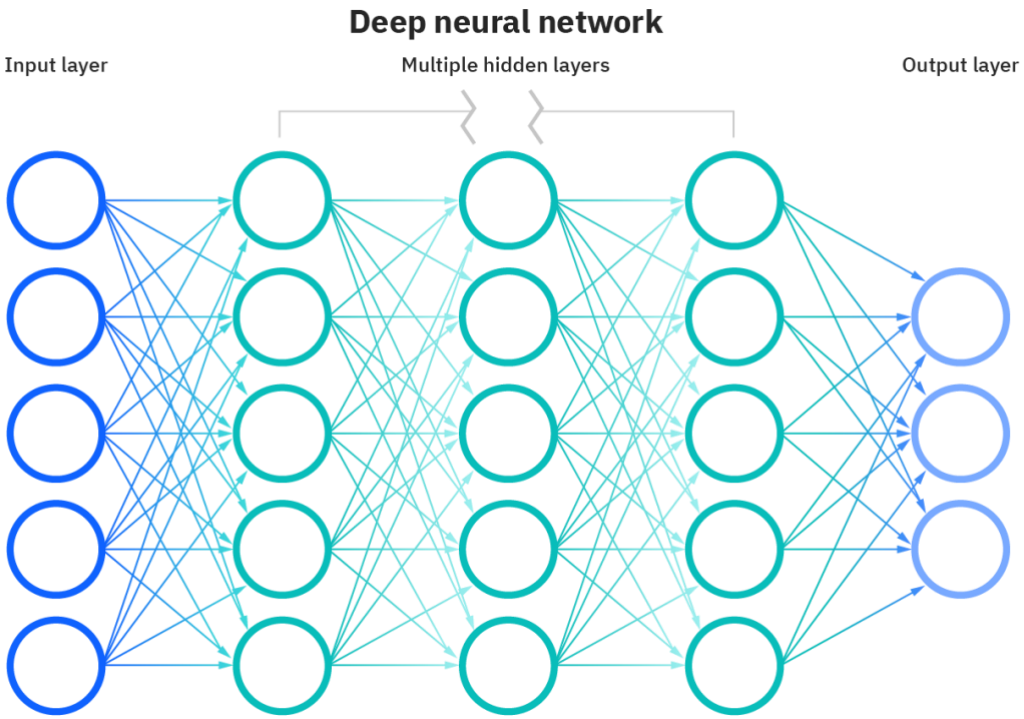



FIGURE 4: STRUCTURE OF FULLY CONNECTED LAYER **[38]**

### 2.4.3 CONVOLUTIONAL NEURAL NETWORKS MODELS

#### 2.4.3.1 LENET-5

LeNet-5 was developed in 1998 by Yan Lecun [39] and is considered to be the pioneer of most modern Convolutional Neural Networks. The model was initially developed to help identify handwritten digits in collaboration with the American postal service to aid in sorting zip codes on mailing envelopes. The LeNet-5 model can be distinguished from its predecessor work by its introduction of convolutional layers. The work produced by Kunihiko Fukushima and Sei Miyake only involved a neural network that consisted of weights and nodes and did not provide a way for feature extraction. The convolutional layers included in Yan Lecun's model provide a way for the network to extract features from images and enable the core functions that are required for image recognition. The convolutional layers included in this model use a subset of the previous layers' channels for each filter to reduce computation complexity and, as a result, break the symmetry within the network. This also allows for subsampling layers to use a form of average pooling, which also reduces the computational complexity. This model is considered to be a classic model and only consists of 60,000 tunable parameters (weights and biases) and can only perform simple tasks such as handwriting recognition. This model was considered to be hindered at the time of creation by the lack of available processing power at the time. The basic architecture of the LeNet-5 model can be seen below in Figure 5.

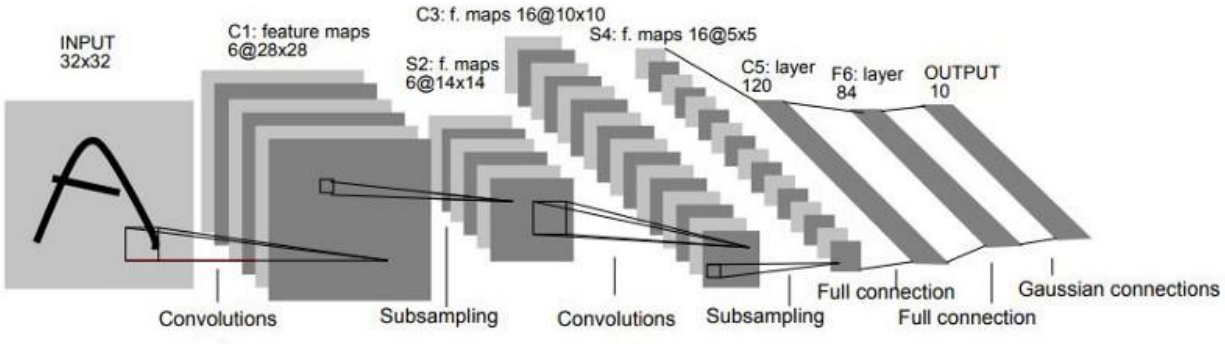



FIGURE 5: ARCHITECTURE OF LENET-5 **[40]**

#### 2.4.3.2 ALEXNET

AlexNet was later developed by Alex Krizhevsky [41]in 2012 to compete in the annual ImageNet competition. AlexNet was inspired by the work completed by Yan Lecun and is considered to be a variant of the LeNet model. The model is considered to be a forefather to Convolutional Neural networks, as, at the time, it demonstrated the functionality of Convolutional Neural Networks. AlexNet is considered to be an incredibly powerful model that is capable of achieving high accuracies on complex datasets. However, AlexNet is a large model that consisted of 60,000,000 tunable parameters and was prone to issues of overfitting. To counter the issue of overfitting, Alex Krizhevsky created the methods of data augmentation and dropout nodes, which were included within the model. Data augmentation involves translating, reflecting images, and rotating the axis of the images to provide to increase the training subset of data. Data augmentation also involves artificially changing the intensities of the RGB channels to dim and brighten images. Alex Krizhevsky also created dropout nodes to assist in removing and deactivating unnecessary neurons. This leads to neurons having more robust features that ensure each neuron serves a purpose. The structure of AlexNet can be seen below in Figure 6.

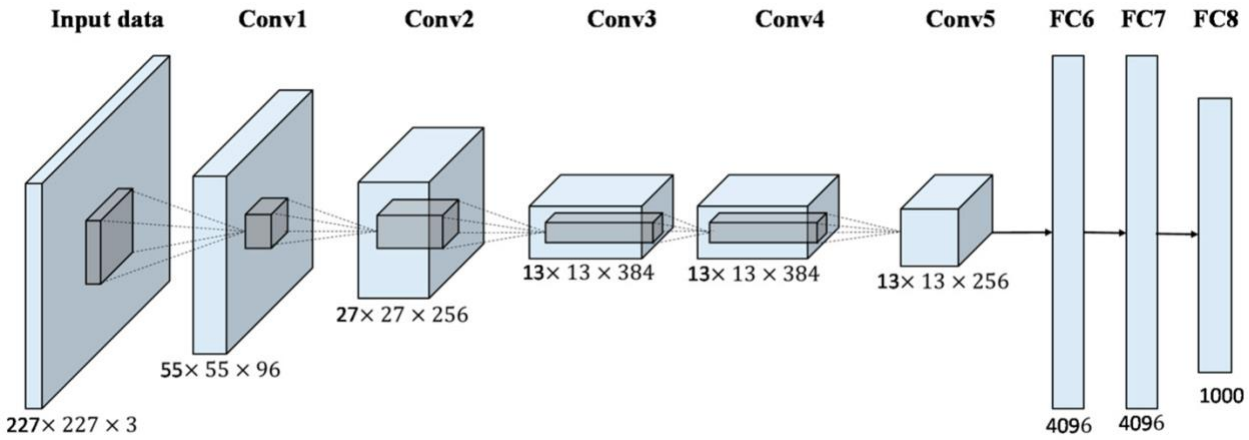



FIGURE 6: THE STRUCTURE OF ALEXNET **[42]**

2.4.3.3 VGG-16

The VGG-16 model offers similar features to the AlexNet model. However, it offers a more straightforward yet more profound architecture. Karen Simonyan and Andrew Zisserman proposed VGG-16 at the University of Oxford in 2013 [43]. This model proposed using a small 3x3 convolutional filter throughout the entire network instead of using large hyper-parameters. The architecture mainly consists of convolutional layers, along with padding and max pool layers of 2x2 filters. The model uses these filters throughout the whole architecture until the final output layer, which consists of 2 fully connected layers. This network has approximately 138 million trainable parameters and is considered to be the modern foundation of most convolutional neural networks. The architecture of the VGG-16 model can be seen below in Figure 7.

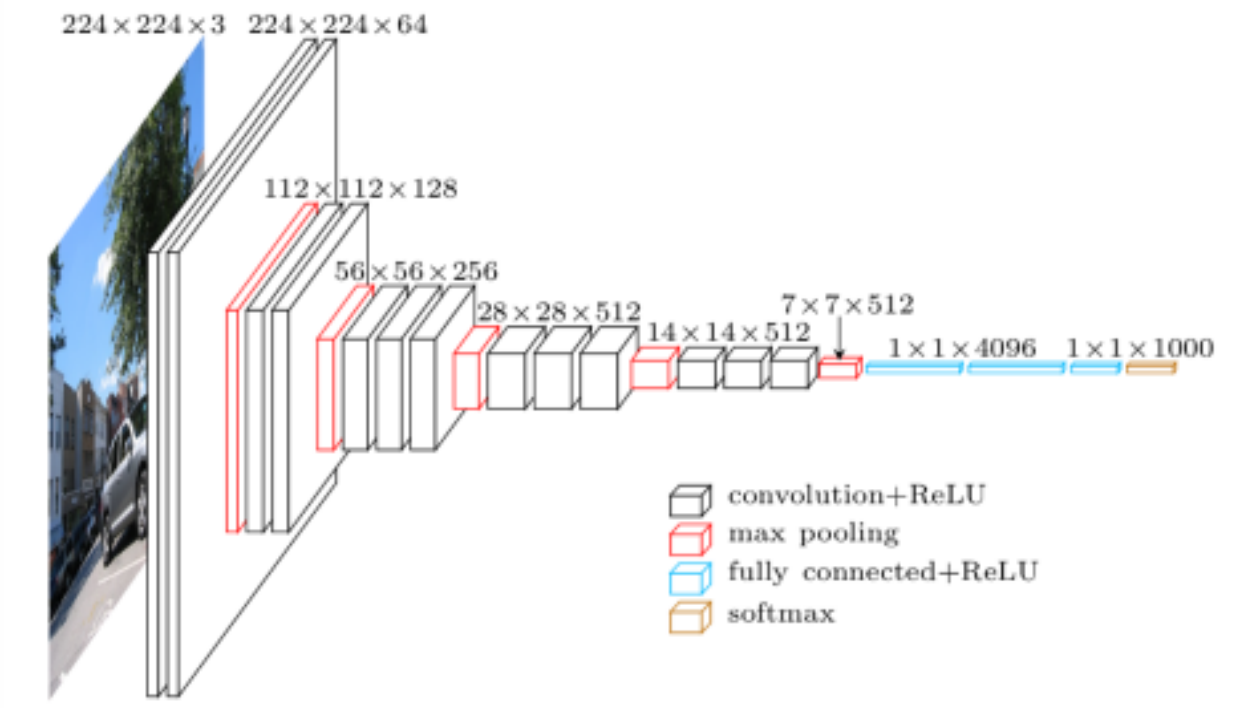



FIGURE 7: ARCHITECTURE OF VGG-16 **[44]**

### 2.4.3.4 INCEPTION (GOOGLENET)

In 2014 researchers at Google introduced the Inception Network, which is an architecture vastly different from VGG-16. The Inception model consists of “Inception Cells”, which perform a series of convolutions of various filter sizes and 5 million trainable parameters. Within an inception cell, multiple convolutions occur on the previous layer with the filter sizes of 1x1, 3x3 and 5x5. This approach can be seen as a robust attempt to use all possible filters to gain the most information from a layer and decrease the loss of information by using an incorrect size filter. This model also incorporates ways to decrease the computational cost of convolutional filters and provide better performance for the model. For example, it was pointed out that 5x5 convolutional can be cheaply replicated by stacking two 3x3 filters. This decreases the variables used when dealing with 5x5 convolutions down from 25 to 18 parameters. The inception model also demonstrated that linear activation functions perform poorly when compared to rectified linear units during all stages of factorization. The architecture of the Incpeption_V1 model can be seen below in Figure 8.

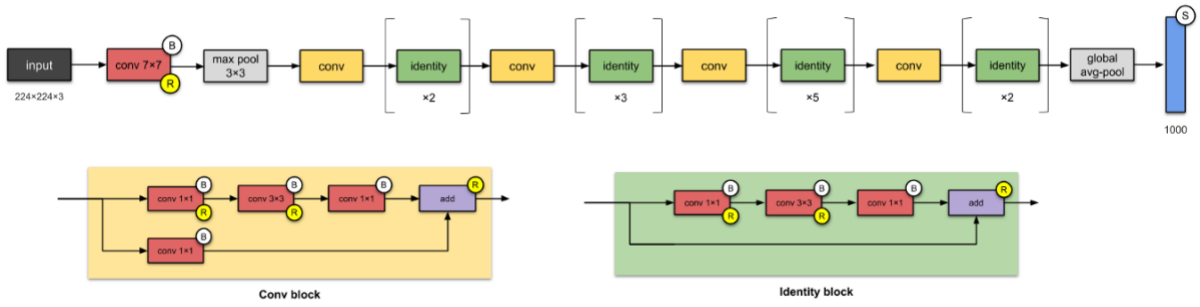



FIGURE 8: ARCHITECTURE OF INCEPTION_V1

#### 2.4.3.5 MOBILENETV1

As model complexity grew, so did the computational power required to run deep neural networks. This led to most deep learning models being restricted to only running on powerful computers with large amounts of video RAM and powerful graphics cards. This resulted in Google developing a class of efficient models called MobileNets [45], which focuses on a low-cost computational model that allows it to run on power-restricted devices such as mobile phones and embedded systems. The focus of MobileNetV1 was to decrease the number of parameters within the model and increase the model's accuracy without compromising its model performance. MobileNetV1 can typically compete with vastly larger models while having a drastically lower number of parameters which results in a decrease in required computational power. MobilenetV1 has achieved this by introducing the Depthwise Separable Convolutions function. The Depthwise Separable Convolutional function is a Depthwise Convolution, followed by a pointwise convolution [46]. A Depthwise Convolution involves using each colour channel of an image as an input instead of merging them together like previous architectures. Once the filter has been applied to each channel, they are stacked together and then a pointwise convolution is applied, condensing the previous stacked colour channels into a flat convolutional layer [47]. This allows for more accuracy to be retained within the architecture while reducing the network's complexity, cost and model size. The architecture for the MobileNet_V1 model can be seen below in figure 9.

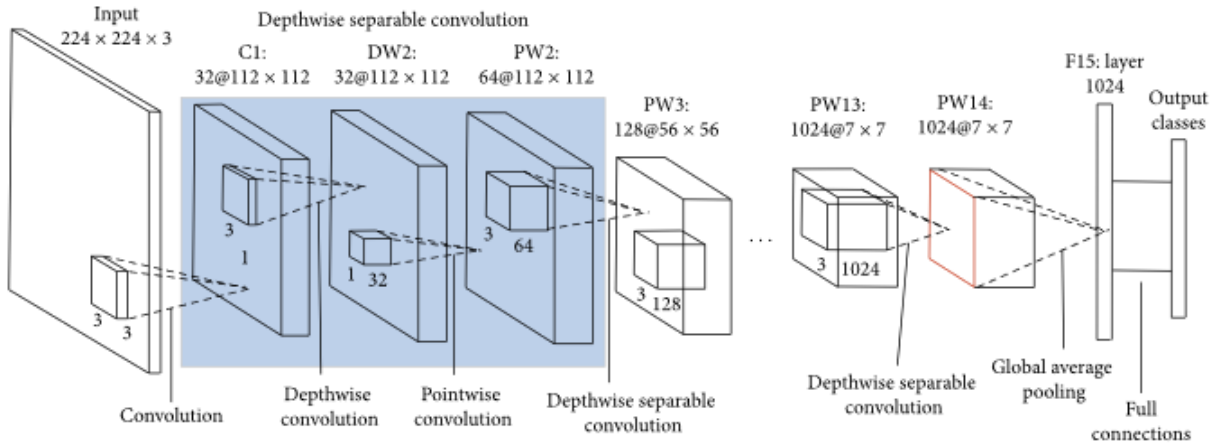



FIGURE 9: MOBILENET_V1 ARCHITECTURE **[48]**

### 2.4.3.6 MOBILENETV2

MobileNetV2 is the successor to the revolutionary MobilNetV1 model. The MobileNetV2 model performs better than the MobileNetV1 model and is suited for mobile and embedded systems applications. The model provides an improved inverted residual structure with the removal of non-linearities in narrow layers. The new model consists of two different blocks. The first block consists of a residual block with a stride of one, and the second is a block with a stride of 2 for downsizing. The first block consists of a 1x1 convolutional filter followed by a 3x3 depthwise block, similar to the depthwise block in MobileNetV1, which is then followed by another 1x1 convolution. This is known as a linear bottleneck and helps reduce the network's computation cost while retaining a vast amount of information. The first block is then fed into the second block, which is similar to the first. However, it uses a stride of 2 for the depthwise convolution. The second block uses a bridge to connect the first blocks together. The architecture for MobileNet_V2 can be seen below in Figure 10.

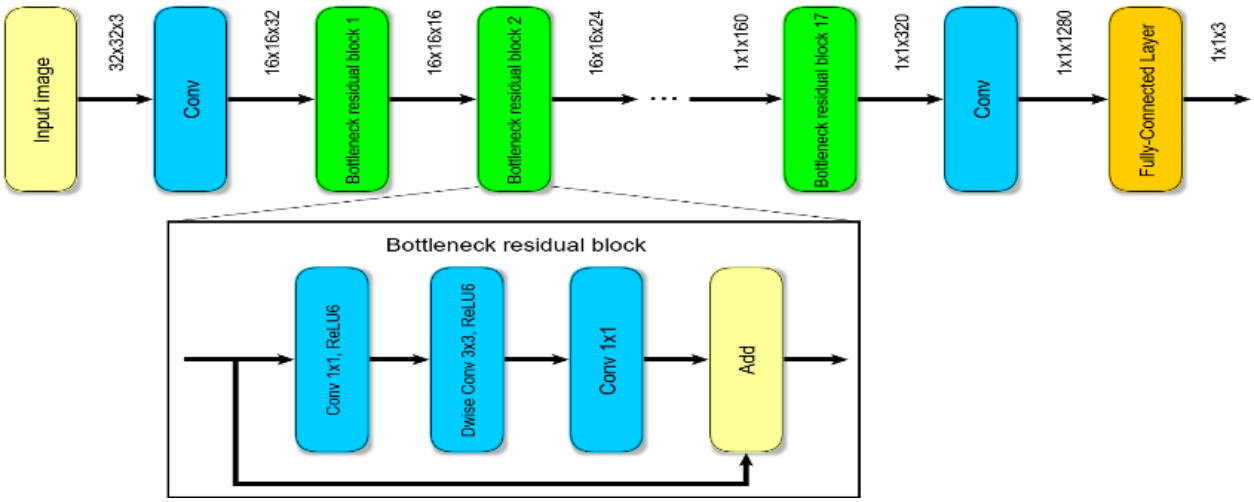



FIGURE 10: MOBILENET_V2 ARCHITECTURE **[49]**

### 2.4.4 ACTIVATION FUNCTIONS

Activation functions are a set of functions that exists between the weights and nodes of the neural network and are considered to be the function that determines whether or not a node will activate or remain dormant. Activation functions are comparable to basic mathematical operations such as a ramp function or a sigmoid curve. When training a Convolutional Neural network, the correct activation function is crucial as it significantly improves performance for a specific task. Each activation function performs differently when actively training a neural network and certain activation functions respond favourably to specific training.

Most Convolutional Neural Networks face a critical problem when training occurs called a vanishing or exploding gradient [50]. The vanishing gradient issue occurs during backpropagation, which is a form of weight adjustment that starts at the output layer and adjusts weights and biases continuously backwards until the original input layer is reached. The algorithm does this to minimise accuracy loss, along with the loss function, which is calculated with respect to each weight in the network [51]. However, during training, the gradient will often dimmish and approach zero, which will eventually leave the weights of the initial or lower layers unchanged. As a result of this, the gradient descent never converges to the optimum, and the training accuracy of Convolutional Neural Networks halts.

The exploding gradient problem is similar to the vanishing gradient problem though it is less common. The exploding gradient continuously increases the training gradient during the backpropagation training algorithm. This causes the weights and biases to become very large, and when weights continuously update and become too large, the gradient descent starts to diverge [52]. These issues are commonly faced during the training of Convolutional Neural Networks, and as a result, different activation functions such as Sigmoid, ReLU, Softsign and Softmax are required to provide the best training for Convolutional Neural Networks. These activation functions and their respective equations are presented in Table 3.

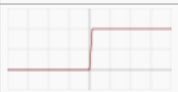
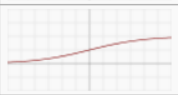
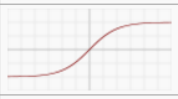
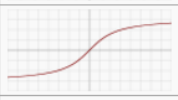
| Name | Plot | Equation | Derivative |
|---|---|---|---|
| Identity |  | $f(x) = x$ | $f'(x) = 1$ |
| Binary step |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$ |
| Logistic (a.k.a Soft step) |  | $f(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)(1-f(x))$ |
| TanH |  | $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ | $f'(x) = 1 - f(x)^2$ |
| ArcTan |  | $f(x) = \tan^{-1}(x)$ | $f'(x) = \frac{1}{x^2+1}$ |
| Rectified Linear Unit (ReLU) |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Parameteric Rectified Linear Unit (PReLU) [2] |  | $f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Exponential Linear Unit (ELU) [3] |  | $f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| SoftPlus |  | $f(x) = \log_e(1 + e^x)$ | $f'(x) = \frac{1}{1+e^{-x}}$ |

TABLE 3: ACTIVATION FUNCTIONS USED IN CONVOLUTIONAL NEURAL NETWORKS **[53]**

#### 2.4.4.1 ReLU

Rectified Linear Unit (ReLU) is considered to be a predominant activation function, which is also known as a ramp function or in a half-wave rectifier in analogue terms. The Rectified Linear Unit first appeared in the 1960s [54] in the context of visual feature extraction by Kunihiko Fukushima [55]. The function was later acclaimed to enable a better training of deep learning networks. The Rectified Linear Unit also provides linear and non-linear variants, such as Leaky ReLU, Softplus and Elu. The Rectified Linear Unit's activation function can be modelled as equation 1 below.

$$f(x) = x_{i,j,k} = \max\left(0, z_{i,j,k}\right) \quad \textbf{(1)}$$

Where $z_{i,j,k}$ is the input activation function at location (*i*, *j*) on the *k*th channel. The Rectified Linear Unit is a piecewise linear function which nulls negative values and retains the values in the positive domain. The max operation allows for ReLU to compute faster than other functions such as the sigmoid or tanh activation functions. The max operation also encourages the network to induce the sparsity in the hidden layers and allows for the network to easily obtain sparse representation.

#### 2.4.4.2 SIGMOID

The Sigmoid function, also known as a logistic or squashing function is a frequently used activation function. The Sigmoid activation function is typically non-linear and commonly appears at the end of deep learning architectures [56]. The Sigmoid function is commonly used in predicting probability-based outcomes and the output layer of a binary classification. The Sigmoid function can be modelled as equation 2 below, where *x* is the input activation function of i. [57]

$$f(x) = \frac{e^{x_i}}{1+e^{x_i}} = \frac{1}{1+e^{-x_i}} \quad \textbf{(2)}$$

The Sigmoid function characteristics provide a function that is considered to be very steep and monotonic. This allows for small changes to drastically impact the overall output when the function is near zero or one. As a result, the output of the function will tend to yield a result that is close to one or zero and the gradient at this region will be diminished, which can be called the "vanishing gradient". When this occurs, the network will experience diminishing returns and will result in a weaker signal being passed through the interconnected nodes.

#### 2.4.4.3 SOFTPLUS

The Softplus activation function is smoothed version of the ReLU activation function. The Softplus function was proposed in 2001 by Duglus et al [58] and was based off the sigmoid activation function. This function features smoothing and nonzero gradient properties that enhance the performance of deep neural networks. The softplus equation can be seen below in equation 3, where *x* is the input activation function of i.

$$f(x) = \log(1 + e^{x_i}) - (1.21) \quad \textbf{(3)}$$

The Softplus activation function generally provides an improved performance increase over typical activation functions such as ReLU and Sigmoid, with increased performance with fewer epochs.

#### 2.4.4.4 SOFTSIGN

The Softsign activation function is vastly different to the ReLU and Sigmoid activation function as the function is based on a quadratic polynomial formula. The Softsign activation function has been commonly used in regression computational problems and is commonly applied to neural language processing. The notable difference between Softsign and the Tanh function is that Softsign converges in polynomial form, whereas tanh converges exponentially. The softsign equation can be seen below in equation 4, where *x* is the input activation function of i.

$$f(x) = \left(\frac{x_i}{|x_i|+1}\right) - (1.13) \quad \textbf{(4)}$$

#### 2.4.4.5 TANH

The Tanh function is also known as the Tangent Hyperbolic Function and is derived from the Sigmoid function. The Tanh activation function consists of a smooth zero centred function that lies between the region of -1 and 1. The Tanh activation function is similar to the Sigmoid and has become the preferred function as it yields increased performance when training deep learning networks. The primary advantage of this function is that it produces an initial zero centred output which in turn aids back-propagation during the training process. The Tanh equation can be seen below in equation 5, where *x* is the input activation function of *i*.

$$\text{Tanh}\, x = \frac{\sinh x}{\cosh x} = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} = \frac{e^{2x_i} - 1}{e^{2x_i} + 1} \quad \textbf{(5)}$$

#### 2.4.4.6 ELU

The Elu activation function is also known as Exponential Linear Unit, which is similar to the ReLU activation function. The Elu activation function typically provides faster learning in deep neural networks along with higher classification accuracy. This is accomplished by avoiding the vanishing gradient problem by setting the starting positive portion of the function to negative. The Exponential Linear Unit also employs a saturated function within the negative region of the function to allow for the decrease of the variation of the unit if deactivated. The Exponential Linear Unit function can be seen below in equation 6, where $\lambda$ is a predefined parameter for controlling the saturation of the negative inputs and where $z_{i,j,k}$ is the input activation function at location ($i$, $j$) on the $k$th channel.

$$f(x) = a_{i,j,k} = \max\left(z_{i,j,k}, 0\right) + \min(\lambda(e^{z_{i,j,k}} - 1), 0) \quad \textbf{(6)}$$

#### 2.4.4.7 SELU

The Selu activation function is also known as the Scaled Exponential Linear Unit which is similar to the RELU activation function and is the scaled version of the Exponential Linear Unit. The Selu activation function does not encounter the vanishing gradient problem and makes it an optimal choice for deep neural networks. The Selu activation functions tends to learn at a much faster rate than most other activation functions and makes it a preferred competitor to the Relu activation function. The Selu equation can be seen below in equation 7, where *x* is the input activation function of i.

$$selu(x) = f(x) = \lambda \begin{cases} x_i, & if\ x > 0 \\ ae^{x_i} - a, & if\ x \leq 0 \end{cases} \quad \textbf{(7)}$$

# CHAPTER 3: METHODOLOGY

## 3.1 INTRODUCTION

The research aims to explore the viability of developing a low-cost system that detects hazardous objects and slipping hazards that will in turn, reduce the fall rate among the elderly and visually impaired. The proposed methodology for the development of the hazard detection system can be demonstrated in sections, the collection of images, training of the convolutional neural network, quantisation of the model and implementation of the hardware.

A dataset will need to be created to train the convolutional neural network. Due to the uniqueness of this study, datasets of liquid spills are not readily available, and none have been made publicly available. As a result, the dataset will be composed of images taken by the researcher. Once a substantial number of images are collected, the images will be labelled and categorised into their respective classes. A custom training algorithm will be required to be written to train the convolutional neural networks by using the TensorFlow and Keras libraries. A custom training algorithm will allow for modifications to be made to the training method to help improve the accuracy of the network. Once the training of the models is complete, the models will then be quantised. Quantisation rescales the weights of the models to 8-bit-integers, allowing them to operate on embedded systems with limited computational cost. Once that has been completed, the models will be transferred to the Google Coral development board, where each model will be evaluated.

## 3.2 DATASET COLLECTION

To train the Convolutional Neural Network a substantial quantity of images will be required. It can be shown that the performance of Convolutional Neural Networks drastically improves when provided with larger datasets to train on [59]. Due to the uniqueness of this study, datasets of liquid spills are not readily available, and none have been made publicly available. As a result, the dataset will be composed of images taken by the researcher. As started in Health Direct Victoria [2] 2 in 3 falls occur within a household environment and can be caused by environmental factors such as spills and poor lighting. The images collected for the dataset will consist of liquid spills on both tiled floorings within a home and carpeted flooring that is commonly found within homes and laminated flooring which is commonly found in kitchens and spill-prone environments.

The lighting within these environments will be varied to increase the accuracy of the Convolutional Neural Network. The images will be collected during the morning, midday and night hours with both natural and artificial lighting being varied. The object of interest will be placed in the foreground, middle ground, and background of an environment, to ensure diversity in the training dataset. Images of the targeted object will be first taken at a distance of 0.3 metres from the target, followed by 0.5 metres, and 1 metre and will continue to increment by 1 metre until the object is no longer visible. Images of the targeted object will also be taken at various angles of 60 degrees to provide the Convolutional Neural Network with enough data to recognise the object from any angle. The diversity in lighting and distance will help to address research aim 1.

To ensure that a variety of images were collected for the dataset, 9 carpet samples were obtained, along with 4 sets of tiles and 3 sets of laminated flooring. These samples were used to help stage spills on various types of backdrops. This was done as spills can occur at a variety of locations with different types of flooring. The carpet samples used within this research were purchased from a carpeting store and have an approximate surface perimeter of 700mm by 1,000mm. While the tiles used in this research have an approximate surface perimeter of 600mm by 900mm. The carpet samples collected were categorised into 4 groups based on the colour and shade of the carpet. This was done due to the fact that the camera used could not distinguish the difference between some of the carpet samples in various lighting conditions. The carpet samples were categorised as dark blue, dark grey, dark brown and light brown/cream. The carpet

samples can be seen below in figure 11, where they are categorised into their respective categories. The tiles and laminated flooring samples that were collected were also categorised together as the Convolutional Neural Network would not be able to tell the difference between laminated flooring and tiled flooring. The categories of tiles used in this research consist of brown tiles, white tiles, and light brown tiles. The tile samples can be seen below in figure 12, where they are categorised into their respective categories. The sample carpet and tiles used within this research do not represent an accurate representation of an indoor environment where a spill can occur, however, the Convolutional Neural Network and training method do not require the training image to be of a full indoor environment. During the labelling of the training set, the bounding boxes will just cover the spill and the convolutional neural network will only learn the details of the spill within the bounding boxes.

FIGURE 11: SAMPLE OF CARPET USED TO CREATE THE DATASET

FIGURE 12: SAMPLES OF TILES USED TO CREATE THE DATASET

The images obtained were taken at a resolution of 1080p by a GoPro HERO 9. The dataset consisted of a total of 11,200 training images and 920 testing images. The spills were conducted on each carpet sample and tiled flooring to ensure a variety of pictures. The liquids used within this experiment were categorised into the following groups, Milk, Orange, Red, Tea, Coffee and water. The liquids that were grouped together consisted of drinks such as black currant juice and raspberry soft drink under the red spill class and tea and coffee under the same class. This was due to the quality of the camera and the lighting within the test environment providing a dull lighting environment, along with the colour of the flooring underneath the spilt liquid making it difficult to distinguish colours clearly. Both datasets also contain photos of the carpet samples and tile samples without liquid spills on them. This subset within the dataset is used as a control group to ensure that the Convolutional Neural Network is not learning the features of the tiles and carpet when detecting the spill. This may lead to a false positive response when the Convolutional Neural Network is operational as it may detect features of similar tiles that were used within the training set. Sample images of spills on tiles can be seen below in figure 13.

FIGURE 13: SPILLS ON TILES

Due to the nature of the carpet, the images collected of spills on the carpet have been categorised by the colour of the carpet and not the colour of the spilled liquid. This is due to the carpet absorbing the liquid spills and darkening, resulting in no colour difference between various spilled liquids. This can be seen below in figure 14, where an orange fizzy drink and tea have been spilt and left for 60 seconds.

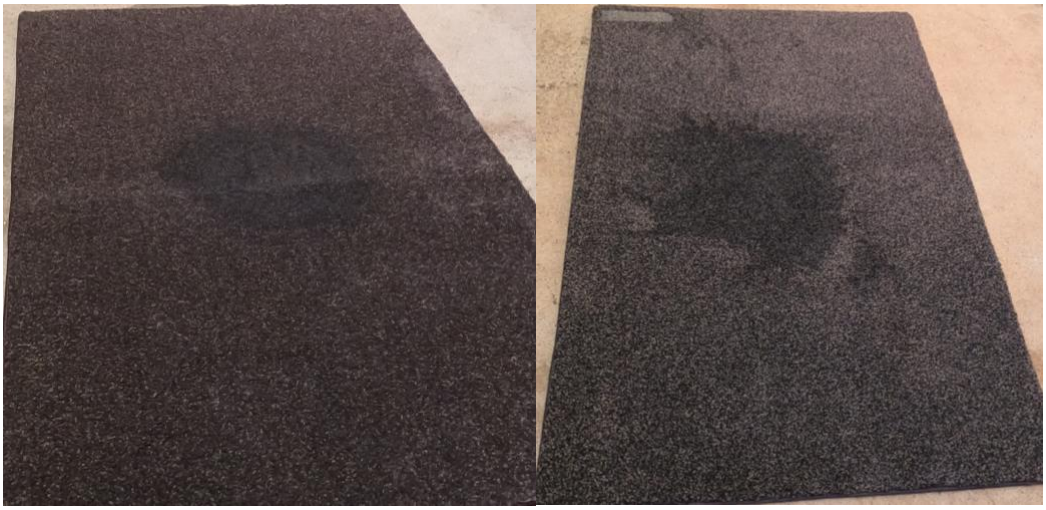


FIGURE 14: DIFFERENCE BETWEEN 2 TYPES OF LIQUIDS SPLIT ON CARPET

A breakdown of the overall training and testing dataset is displayed in table 4.

| Training | | Testing | |
|---|---|---|---|
| Images Total | 11,200 | Images Total | 920 |
| | | | |
| Spills on Tiles | | Spills on Tiles | |
| Milk | 940 | Milk | 80 |
| Orange | 860 | Orange | 60 |
| Red | 980 | Red | 70 |
| Coffee/Tea | 770 | Coffee/Tea | 60 |
| Water | 800 | Water | 90 |
| Total | 4,350 | Total | 360 |
| | | | |
| Tiles with no spills | | Tiles with no spills | |
| White | 130 | White | 30 |
| Brown | 210 | Brown | 40 |
| Light | 160 | Light | 30 |
| Total | 600 | Total | 100 |
| | | | |
| Spills on Carpet | | Spills on Carpet | |
| Cream | 680 | Cream | 60 |
| Light Grey | 1590 | Light Grey | 90 |
| Navy | 500 | Navy | 70 |
| Dark Grey | 970 | Dark Grey | 80 |
| Total | 3,740 | Total | 300 |
| | | | |
| Carpet no Spills | | Carpet no Spills | |
| Cream | 450 | Cream | 40 |
| Light Grey | 790 | Light Grey | 40 |
| Navy | 300 | Navy | 30 |
| Dark Grey | 990 | Dark Grey | 50 |
| Total | 2,530 | Total | 160 |

TABLE 4: CONTENTS OF TRAINING AND TESTING DATASET

Once the images have been obtained for the dataset, they are required to be converted into a readily usable image format. The image format PNG was chosen as it provides image compression with minimal losses or artifacts. The file size of Images is important as the training process involves loading sets of images onto a graphics card of a computer and is limited by the size of the video memory available on the graphics card. It is ideal that the entirety of the training dataset can be loaded onto the graphics card as this can drastically hinder the training performance and lead to memory issues whilst training is occurring. Once the images are converted to the correct format, it is recommended that they are resized to fit the application's input. In this case, the Google Coral Development boards camera will be operating at 720p resolution. This choice is mainly due to preserving power consumption on the board, along with helping to decrease the complexity of the model. The development board's camera is native to 1080p, however running the model at 1080p will require the processing of 2.1 million pixels compared to 0.92 million pixels when operating at 720p. This option helps to reduce the computational cost by about half when both options are compared.

Once the formatting of images has been completed, the images will be required to be sorted and labelled. The images will be sorted into subdirectories within the training dataset, this is done for ease of use when altering the training dataset for the model. The images will then be loaded into a labelling software named “LabelImg”. This program is a Python script with a graphical user interface that allows the user to draw binding boxes around the targeted object within the image and allows it to include the background information within the image, such as other objects. This method was chosen as unnecessary information will be contained within the images and will help to clear confusion within the model. The software produces an XML file that contains the pixel coordinates of the object within each image, along with the label of the focused object. This data produced within the XML files were then grouped and converted into a CSV file. The CSV file is used within the training and holds an overview of all the data for each image.

When the processing and labelling of the datasets were complete, they were loaded into the Python script. The training, testing and validation datasets were loaded into the Python environment by using a Keras pre-processing function. This method allowed for certain variables to be set, such as the shuffle, seed, batch size and validation split. These variables determine the quantity and load order of the images used for training and the portion of images that are

separated for the validation dataset. The code used to load the datasets can be seen below in figure 15.

```
ds_train = tf.keras.preprocessing.image_dataset_from_directory(
    r"C:\Users\PycharmProjects\CNN\Data\Train",
    labels="inferred",
    label_mode="int",
    color_mode="rgb",
    batch_size=batch_size,
    image_size=(img_height, img_width),
    shuffle=True,
    seed=123,
    validation_split=0.1,
    subset="training",)
```

FIGURE 15: SAMPLE CODE DEMONSTRATING THE LOADING OF DATASETS

## 3.3 CONVOLUTIONAL NEURAL NETWORK TRAINING

### 3.3.1 INSTALLATION OF LIBRARIES AND SOFTWARE

The training of the Convolutional Neural Network will be undertaken in Python using the TensorFlow and Keras libraries. TensorFlow is an open-source software library written for the training and development of machine learning models and artificial intelligence models. TensorFlow was chosen for this project as it was developed by the Google Brain team which is a subdivision of Google AI [60]. Google AI is a division of Google that focuses on the research and development of artificial intelligence and is responsible for the development of the Tensor Processing Unit and various machine learning architectures. The Google Coral Development board is also considered to be more reliable and streamlined for the use of TensorFlow models, in comparison to its competitor PyTorch [61].

Keras is a Python API that provides a method of interfacing with the TensorFlow library [62]. This API library is used as an ease-of-use function as it provides the ability to easily implement building blocks of neural networks. Keras allows for the implementation of pre-built models, along with the ability to create custom models, layers, objectives, activation functions and optimizers. Keras also provides various testing functions that can be performed on Convolutional Neural Networks.

The TensorFlow Library and Keras library will be installed through the use of Anaconda [63] which is a Python-based environment that specialises in the easy installation of Python packages and software and allows for the creation of virtual environments that allow for different versions of TensorFlow and Keras to function. Anaconda is vital for this project as it allows for the virtual environments to be accessed through PyCharm [64] which is a Python graphical user interface similar to MATLAB that will be used throughout the research as a means to write the training and testing scripts required for the research. A workflow diagram can be seen below in figure 16


FIGURE 16: THE WORKFLOW DIAGRAM OF TRAINING METHOD

### 3.2.2 CREATION OF CONVOLUTIONAL NEURAL NETWORK MODELS

Once the software and libraries are installed a custom script is required to be written to set the training and environment variables along with the importation of libraries into PyCharm. The images collected in the previous section are then required to be loaded into the PyCharm environment. The chosen Convolutional Neural Network will be required to be built from scratch and not imported as a precomposed TensorFlow model. The model will be required to be built from scratch as it allows for the convolutional layers and activation functions to be altered and removed or added to gain better accuracy across the model. The models selected mainly consist of various convolutional layers that consisted of both regular convolutional layers and separable convolutional layers. Keras allows for each convolutional layer to be defined, allowing for the parameters to be altered. The standard convolution layer consists of three main parameters, The layer, batch normalization and the activation function. The layer parameter allows for the definition of the kernel size, the stride length, and the size of the padding. Whereas the batch normalization layer enables the current layer to normalize the input received from the previous layer. This increases the layer's ability to learn independently in an isolated environment. The activation function is a predetermined function that is responsible for the discission to pass information onto the next layer. An example code has been taken from the convolutional layers of the MobileNet_V1 architecture that demonstrates the convolutional layer and the separable convolutional layer. This code can be seen below in figure 19.

```
Def Conv( x , num_filters , kernel_size , strides=1 , alpha=1.0
):
x = tf.keras.layers.Conv2D( np.floor( num_filters * alpha ) ,
kernel_size=kernel_size , strides=strides , use_bias=False ,
padding='same' )( x )
x = tf.keras.layers.BatchNormalization( momentum=0.9997 )(x)
x = tf.keras.layers.Activation('relu')(x)
return x

def SeparableConv( x , num_filters , strides , alpha=1.0 ):
x = tf.keras.layers.DepthwiseConv2D( kernel_size=3 ,
padding='same' )( x )
x = tf.keras.layers.BatchNormalization(momentum=0.9997)( x )
x = tf.keras.layers.Activation( 'relu' )( x )
x = tf.keras.layers.Conv2D( np.floor( num_filters * alpha ) ,
kernel_size=( 1 , 1 ) , strides=strides , use_bias=False ,
padding='same' )( x )
x = tf.keras.layers.BatchNormalization(momentum=0.9997)(x)
x = tf.keras.layers.Activation('relu')(x)
return x
```

FIGURE 17: THE CODE DEFINITION OF CONVOLUTIONAL LAYERS

The architecture of the selected models can be replicated by first setting the desired parameters of each model during the defining section of the code. Once the convolutional layers have been defined, they can be stacked in order to create the various layers of the overall architecture. A sample of code has been taken from the MobileNet_V1 model that demonstrates the input layer along with the convolutional layers and the separable convolutional layers. The sample code can be seen in figure 20 below. The input layer is responsible for scaling the input images to the correct size of the convolutional neural network, along with defining the number of colour channels within the image. The sample code also demonstrates the flattening of the last convolutional layer, which prepares the architecture to enter the fully connected portion of the model. The fully connected layer is then connected to the last binary classification layer, which determines the probability of correctly labelling an object.

```
inputs = tf.keras.layers.Input( shape=( 720 , 1280 , 3 ) )
x = Conv( input, num_filters=32 , kernel_size=32 , strides=2 )
x = SeparableConv( x , num_filters=32 , strides=1 )
x = Conv( x , num_filters=64 , kernel_size=1 )
x = SeparableConv( x , num_filters=64 , strides=2  )
```

```
x = tf.keras.layers.AveragePooling2D( pool_size=( 7 , 7 ) )( x
)
x = tf.keras.layers.Flatten()( x )
x = tf.keras.layers.Dense( Num of classes )( x )
```

FIGURE 18: SAMPLE CODE OF THE ARCHITECTURE OF MOBILENET_V1

The Convolutional Neural Network models investigated for this research was the MobileNet V1, MobileNet V2, Inception V3 and VGG-16. MobileNetV1 was selected as this architecture is designed to be a lightweight architecture that is considered to be viable to run on systems with limited computational power and limited battery life. MobileNet V2 is the successor to the MobileNet V1 architecture and has been suggested to have improved accuracy and performance over the MobileNet V1 model. A table showing the comparison between each model and their architecture can be seen below in table

| Model | Layers | Parameters | Kernel size | Year of release |
|---|---|---|---|---|
| MobileNet V1 | 28 | 4.2 million | 3x3 | 2017 |
| MobileNet V2 | 53 | 3.4 million | 3x3 | 2018 |
| Inception V3 | 48 | 23 million | 1x1, 3x3, 5x5 | 2015 |
| VGG-16 | 16 | 138 million | 3x3 | 2018 |

TABLE 5: COMPARISON OF THE SELECTED MODELS

The architecture of the selected models can be seen below in tables 6 to 9.

Table 1. MobileNet Body Architecture

| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | $3 \times 3 \times 3 \times 32$ | $224 \times 224 \times 3$ |
| Conv dw / s1 | $3 \times 3 \times 32$ dw | $112 \times 112 \times 32$ |
| Conv / s1 | $1 \times 1 \times 32 \times 64$ | $112 \times 112 \times 32$ |
| Conv dw / s2 | $3 \times 3 \times 64$ dw | $112 \times 112 \times 64$ |
| Conv / s1 | $1 \times 1 \times 64 \times 128$ | $56 \times 56 \times 64$ |
| Conv dw / s1 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 128$ | $56 \times 56 \times 128$ |
| Conv dw / s2 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 256$ | $28 \times 28 \times 128$ |
| Conv dw / s1 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 256$ | $28 \times 28 \times 256$ |
| Conv dw / s2 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 512$ | $14 \times 14 \times 256$ |
| $5\times$ Conv dw / s1 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| $5\times$ Conv / s1 | $1 \times 1 \times 512 \times 512$ | $14 \times 14 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| Conv / s1 | $1 \times 1 \times 512 \times 1024$ | $7 \times 7 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 1024$ dw | $7 \times 7 \times 1024$ |
| Conv / s1 | $1 \times 1 \times 1024 \times 1024$ | $7 \times 7 \times 1024$ |
| Avg Pool / s1 | Pool $7 \times 7$ | $7 \times 7 \times 1024$ |
| FC / s1 | $1024 \times 1000$ | $1 \times 1 \times 1024$ |
| Softmax / s1 | Classifier | $1 \times 1 \times 1000$ |

TABLE 6: ARCHITECTURE OF MOBILENET_V1 **[65]**

| Input | Operator | $t$ | $c$ | $n$ | $s$ |
|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 32 | 1 | 2 |
| $112^2 \times 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 \times 16$ | bottleneck | 6 | 24 | 2 | 2 |
| $56^2 \times 24$ | bottleneck | 6 | 32 | 3 | 2 |
| $28^2 \times 32$ | bottleneck | 6 | 64 | 4 | 2 |
| $14^2 \times 64$ | bottleneck | 6 | 96 | 3 | 1 |
| $14^2 \times 96$ | bottleneck | 6 | 160 | 3 | 2 |
| $7^2 \times 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 \times 320$ | conv2d 1x1 | - | 1280 | 1 | 1 |
| $7^2 \times 1280$ | avgpool 7x7 | - | - | 1 | - |
| $1 \times 1 \times 1280$ | conv2d 1x1 | - | k | - | |

TABLE 7: ARCHITECTURE OF MOBILENET_V2 **[66]**

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input ($224 \times 224$ RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

TABLE 8: ARCHITECTURE OF VGG-16 **[67]**

| **type** | **patch size/stride** or remarks | **input size** |
|---|---|---|
| conv | $3\times3/2$ | $299\times299\times3$ |
| conv | $3\times3/1$ | $149\times149\times32$ |
| conv padded | $3\times3/1$ | $147\times147\times32$ |
| pool | $3\times3/2$ | $147\times147\times64$ |
| conv | $3\times3/1$ | $73\times73\times64$ |
| conv | $3\times3/2$ | $71\times71\times80$ |
| conv | $3\times3/1$ | $35\times35\times192$ |
| $3\times$Inception | As in figure 5 | $35\times35\times288$ |
| $5\times$Inception | As in figure 6 | $17\times17\times768$ |
| $2\times$Inception | As in figure 7 | $8\times8\times1280$ |
| pool | $8 \times 8$ | $8 \times 8 \times 2048$ |
| linear | logits | $1 \times 1 \times 2048$ |
| softmax | classifier | $1 \times 1 \times 1000$ |

TABLE 9: ARCHITECTURE OF INCEPTION_V3 **[68]**

### 3.2.3 TRAINING METHOD

Once the dataset and model are loaded into the Python script a custom training algorithm is written as this will increase performance over a standard training algorithm provided by TensorFlow. The custom training algorithm is vital to this research as it will drastically increase the accuracy of the Convolutional Neural Networks and help prevent against the vanishing gradient problem. TensorFlow and Keras provide a pre-compiled Model Fit function which acts as the basic training function in TensorFlow. The Model Fit function does not allow for any alterations and is locked into a preconfigured training method. This preconfigured training method is prone to the vanishing gradient problem and requires either a larger training dataset or a custom training algorithm to overcome this issue. The standard TensorFlow testing algorithm will then be used to evaluate the model. A workflow diagram can be seen below in figure 19.

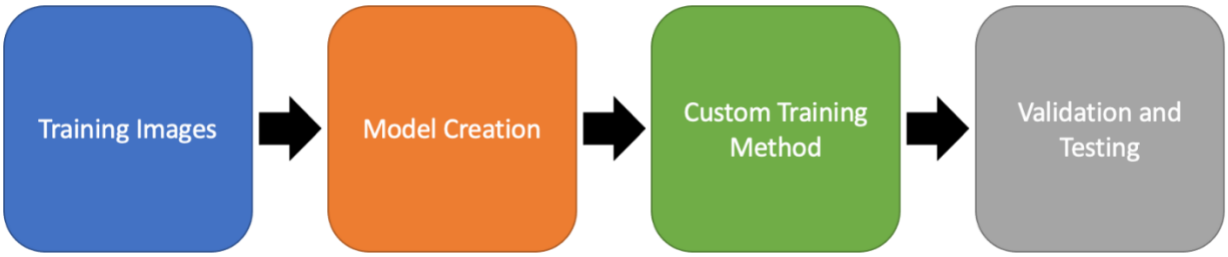



FIGURE 19: TRAINING WORKFLOW

The custom training algorithm will also take advantage of Data Augmentation. Data Augmentation is a method used in machine learning that purposely skews and alters data to increase the size of the training dataset. This is achieved through methods such as flipping images, varying the brightness, and rotating the axis of images. The Data Augmentation function can be seen below in figure 20.

```
def augment(x, y):
    image = tf.image.random_brightness(x, max_delta=0.05)
    return image, y
```

FIGURE 20: THE SAMPLE CODE USED FOR DATA AUGMENTATION

The custom training algorithm first begins by defining and loading the selected model into the training algorithm. The script then defines the optimizers used for the training of the architecture and defines the method of training. The sample code used to train the models is provided in figure 21. It can be seen in the code below that the training will operate at each step on the accuracy loss method. The script then defines how the model's weights will be adjusted for each step and how the gradient will be adjusted accordingly.

```
class CustomFit(keras.Model):
    def __init__(self, model):
        super(CustomFit, self).__init__()
        self.model = model

    def compile(self, optimizer, loss):
        super(CustomFit, self).compile()
        self.optimizer = optimizer
        self.loss = loss

    def train_step(self, data):
        x, y = data

             with tf.GradientTape() as tape:
                 y_pred = self.model(x, training=True)
            loss = self.loss(y, y_pred)
        training_vars = self.trainable_variables
             gradients = tape.gradient(loss, training_vars)
        self.optimizer.apply_gradients(zip(gradients,
training_vars))
             acc_metric.update_state(y, y_pred)
        return {"loss": loss, "accuracy": acc_metric.result()}
```

FIGURE 21: THE SAMPLE CODE OF TRAINING

This method of training has provided positive results, as preliminary testing showed an average of a 15% improvement when training convolutional neural network models. The custom training method has also reduced the occurrence of the vanishing gradient problem. The standard TensorFlow training method encountered the vanishing gradient problem when the training of the model reached 70-80% accuracy. The custom training algorithm overcame this issue and allowed for models to train upwards of 90% accuracy.

### 3.2.4 TESTING METRICS

The standard TensorFlow Testing algorithm was used to evaluate each model. This method was selected as it reduced the bias in testing and helped to prevent inaccurate results due to coding errors. The standard TensorFlow method allows for the observation of 4 main metrics. The primary metric is the accuracy, precision, recall and F1 score. The equations for each of these metrics can be seen below in equations 8 to 11.

$$Precision = \frac{TP}{TP+FP} \quad \textbf{(8)}$$

$$Recall = \frac{TP}{TP+FN} \quad \textbf{(9)}$$

$$F1\ score = 2 * \frac{Precision*Recall}{Presion+Recall} \quad \textbf{(10)}$$

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \quad \textbf{(11)}$$

The True Positive (TP) and True Negative (TN) metrics refer to the number of objects that were correctly identified and correctly negatively identified, while the False Negative (FN) refers to the objects that were not detected by the model. While the metric False Positive (FP), refers to the number of objects that were falsely identified. Therefore, based on equation 5.1 the precision metric is the ratio of objects that have been positively identified compared to the rate of falsely identified objects. The recall metric measures the ratio of correctly detected objects compared to the baseline or ground truth in the training dataset and is defined in equation 5.2. The accuracy and F1 score are commonly used metrics to evaluate the performance of convolutional neural networks. While the accuracy metric is a widely used metric which can be used to summarize the performance of the model, which takes into consideration the true positive and negative metrics, while the F1 score takes into consideration the False positive and negative metrics.

## 3.4 QUANTISATION

The fundamental idea behind quantization is the conversion of weights and inputs to integer-based data. Traditionally machine learning models, especially convolutional neural networks are based on each weight being a 64-bit floating-point data type. Quantisation converts these 64-bit floating-point data types to 8-bit integers. Quantisation essentially reduces the model's complexity and computational cost to a quarter of its original cost. Quantisation also allows for models to be able to operate on computational restricted devices, such as smartphones or embedded system platforms. The quantisation of models is required for the models to run on the Google Coral Development Board. Once the selected models were trained in the previous section a new training algorithm was created. The new training algorithm was based upon the quantised method of Post-training Quantisation. This method allows for the quantisation of pre-trained convolutional network models. The training algorithm from the previous section was used with the addition of a quantisation section. The quantisation section consisted of the code shown below in figure 22. This code converts the already trained TensorFlow model into a TF lite model which is then retained with the new 8-bit integer weights.

```
converter =
tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()
```

FIGURE 22: SAMPLE CODE FOR MODEL CONVERSION

## 3.5 INTEGRATION OF HARDWARE

The Google Coral Development board is a single board computer similar to the Raspberry Pi that hosts a tensor processing unit. To transfer the TensorFlow Lite models across the Google Coral Board along with the Python script used to run the models requires the use of the Mendel Development Tool. The Mendel Development Tool is a command-line tool that allows for interfacing between the Mendel operating system on the Google Coral Development Board and a regular computer. Files can be pushed between the computer and Google Coral development Board by the use of the push and pull command. To run the TensorFlow Lite model a custom script needs to be written that defines the source video input, location of the model and draws bounding boxes and recorded accuracy within the model. The source input of the model is provided by the use of the Google Coral Camera. A workflow diagram can be seen below in figure 23.

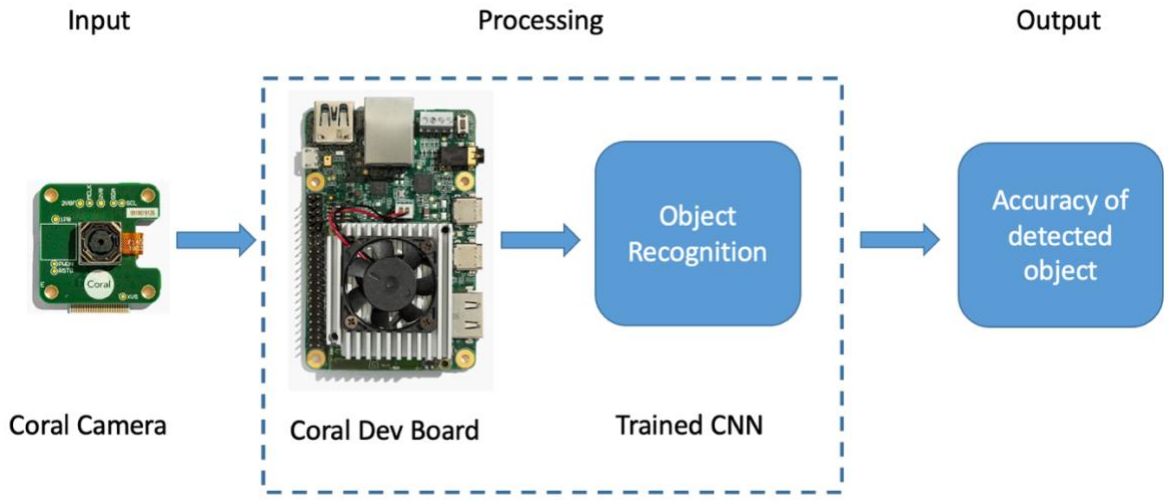



FIGURE 23: WORKFLOW DIAGRAM OF GOOGLE CORAL DEVELOPMENT BOARD

An experimental housing unit will also be designed for the Google Coral Development Board as means to complete a static test in chapters 4 to 6. The housing unit will be designed using Tinkercad and will be created using a 3D printer. A prototype of the housing can be seen below in figure 24. The housing will include mounting points for the Google Coral Camera, along with a mounting system that can be used to mount the prototype onto an

individual. The prototype will be tested in various positions on an individual with the primary target area being mounted to a belt or strap to hold the prototype in place. The prototype case consists of the dimensions 9.5cm x 6.5cm x 3cm and provides venting on the sides of the case and front of the case to allow for airflow to the board to assist in cooling. The case also features a rudimental back cut out to allow for a belt with a width of 5.5cm to fit through the case.

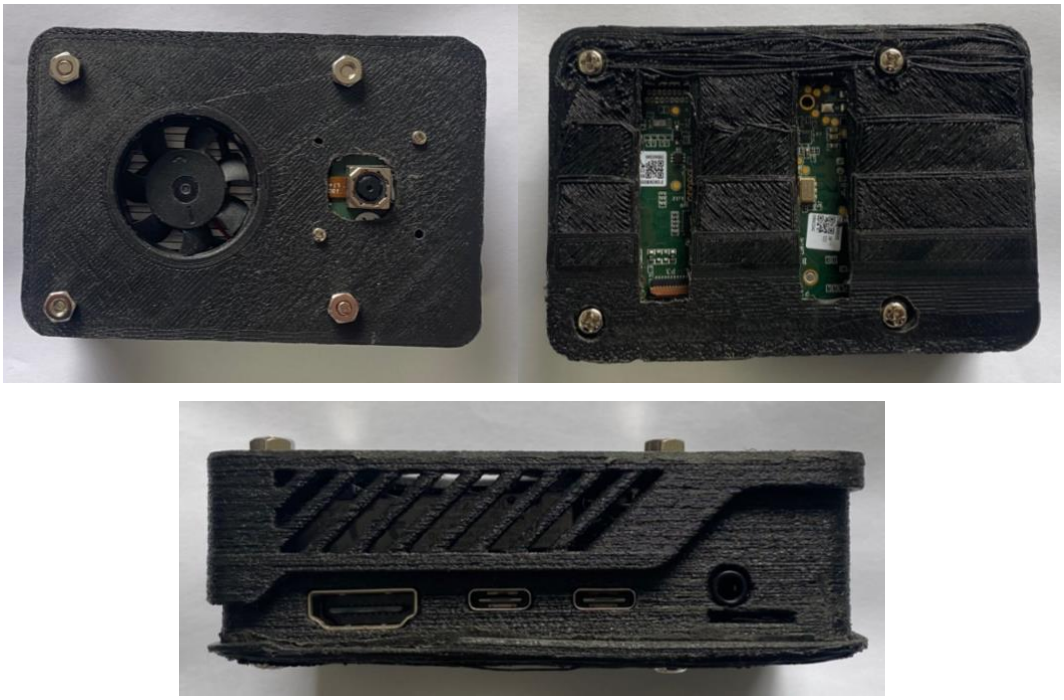


FIGURE 24: PROTOTYPE HOUSING OF THE GOOGLE CORAL DEVELOPMENT BOARD

Chapters 4 to 6 will include a static trial where the system will be placed at a static location and a spill will be placed in front of the system. The system will be placed on a tripod and the distance from the object to the tripod will be changed at 1-metre intervals, starting at 0.5 metres. The object will be moved to various locations to ensure a change in the background and environment. The lighting will be varied to gauge the robustness of the model, along with other objects being placed in front or behind the target object to obscure the prototype's vision. The variations within the lighting conditions and distance of the focused object will help to address the first aim of the study. The model will also provide a text file of the lowest, highest, and mean accuracy of the detected object.

# CHAPTER 4: SPILLS DETECTION ON TILES USING CONVOLUTIONAL NEURAL NETWORKS

## ABSTRACT

This study determines whether it is possible to detect liquid spills on tiled and laminated flooring with the use of a non-linear machine learning model such as a Convolutional Neural Network. It was hypothesised that a convolutional network will be able to detect spills on tiled surfaces to a high degree of accuracy. To achieve this high degree of accuracy, multiple convolutional neural network models will be tested, such as MobileNet_V1, MobileNet_V2, Inception_V3 and VGG-16. The listed models will be trained on a dataset containing 5,410 images of spills that can be broken down into 8 sets of classes. The spilled liquids have been categorised into 5 classes: Milk, Orange, Red, Coffee/Tea and Water. The training data also contains 3 classes of tiles containing no spills as a control group for the experiment. The classes with no spills consist of White, Brown, and light, with the light category containing a variety of cream and light brown coloured tiles. The models will also be tested on a testing dataset of 460 images that are categorised into the 8 classes listed above. The models will also be quantised and evaluated as quantising a model will reduce the accuracy of each model. The quantised models will be evaluated on the Google Coral Development board as this will ensure that a developed model can operate on an embedded systems platform when in a simulated environment. The models that are under evaluation will be examined using the following metrics of analysis, precision, recall, F1 score and accuracy. These metrics allow for comprehensive observation of how the different architectures performed under the same training and testing environment. The MobileNet_V1 model outperforms the newer MobileNet_V2, VGG-16 and Inception_V3. It can be seen that the other models suffer from overfitting as a result of their complex architecture.

## 4.1 INTRODUCTION

As started in Health Direct Victoria [2] an average of 1 in 3 elderly Australians will experience a serious fall every 12 months, with 1 in 5 of these falls requiring hospital attention. It is also stated that 2 in 3 of these falls occur within a household environment and can be caused by environmental factors such as spills and poor lighting. It is also stated that falls are more likely to occur on hard surfaces such as tiles and laminated flooring, as these materials do not possess the ability to absorb various types of liquids and can become drastically slippery when wet. Hence the importance to develop a mobile aid device that can notify the elderly individual to hazardous liquid spills. There is currently a lack of mobile aid devices that use computer vision technology to aid in the assistance of the vision impaired and elderly community. Many previous devices in the form of walking canes and wearable sensors have taken advantage of machine learning algorithms in an attempt too pre-emptively predict falls of elderly individuals. These devices have used rudimentary machine learning models such as neural networks or support vector machine algorithms. Previous approaches have also applied traditional machine learning algorithms to assist in identifying objects for the visionally impaired, such as in the case of the OrCam [7]. However, the OrCam does not use convolutional neural networks on a live video feed and instead interprets a single image taken by the user. Convolutional Neural network are a recent concept to the artificial intelligence industry and have provided many advantages in the computer vision field and are predominately used for live video object detection or image segmentation.

This chapter aims to determine whether it is possible to detect liquid spills on tiled and laminated flooring with the use of a non-linear machine learning model such as a Convolutional Neural Network. To achieve this high degree of accuracy, multiple convolutional neural network models will be tested, such as MobileNet_V1, MobileNet_V2, Inception_V3 and VGG-16. The listed models will be trained on a dataset containing 5,410 images of spills that can be broken down into 8 sets of classes. The spilled liquids have been categorised into 5 classes: Milk, Orange, Red, Coffee/Tea and Water. The training data also contains 3 classes of tiles containing no spills as a control group for the experiment. The classes with no spills consist

of White, Brown, and light, with the light category containing a variety of cream and light brown coloured tiles.

## 4.2 MODEL COMPARISON

The Convolutional Neural Networks architectures that will be evaluated during the research consist of MobileNet_V1, MobileNet_V2, Inception V3 and VGG-16. It is hypothesised that the MobileNet_V1 architecture will be the predominant architecture among the competitors as the architecture can be considered to be simpler and more primitive when compared to a model such as Inception_V3. Less complex models tend to perform better when trained to recognise a limited number of classes, when compared to the more complex models.

The MobileNet_V1 and V2 models have a relatively small number of tuneable paraments when compared to the inception_V3 and VGG-16 models. MobileNet_v1 and MobileNet_V2 consist of 4.2 million and 3.4 million parameters respectively. Whereas Inception_V3 and VGG-16 have 23 million and 138 million respectively. The larger and more complex models such as Inception_V3 and VGG-16 are more likely to experience overfitting as their models will start to memorise training images instead of learning the key features in them. The VGG-16, MobileNet_V1 and V2 each have a kernel filter size of 3x3, meaning that each model should be capable of learning the features of spilled liquids, whereas the Inception_V3 model has multiple kernel filter sizes of 1x1, 3x3 and 5x5. These kernel filters may pose a disadvantage to the architecture as the 1x1 filter maybe too small to accurately detect the edges of spills and the 5x5 kernel filters may be too large and result in edges not being detected properly. A table summarising the models can be seen in chapter 3.2.2 in table 3.

## 4.3 METHODS

### 4.3.1 TRAINING DATASET

The training and testing dataset used for this section of the research consisted of 5,410 images with dimensions of 1280x720 pixels. The images were taken at various distances from the targeted spill, ranging from 0.30 metres to 4 meters. The images were also taken from various angles on both the vertical and horizontal axis, typically moving around the spill in 60-degree increments. The images were obtained by using a go pro hero 9 and were taken at various times throughout the day to ensure a variety of lighting conditions.

The dataset used to train the Convolutional Neural Networks contains 8 sets of classes. The spilled liquids have been categorised into 5 classes: Milk, Orange, Red, Coffee/Tea and Water. Each one of these classes contains various spilled drinks but has been grouped together as the camera used within this experiment could not distinguish the difference between some types of liquids. The liquids that were grouped together consisted of drinks such as black currant juice and raspberry soft drink under the red spill class and tea and coffee under the same class. This was due to the quality of the camera and the lighting within the test environment providing a dull lighting environment, along with the colour of the flooring underneath the spilt liquid making it difficult to distinguish colours clearly.

The dataset selected also consists of a subset of classes that are going to be used as a control class to ensure that the Convolutional Neural Network is not learning the features of the tiles when detecting the spill. These control classes consisted of white, brown, and cream tiles. It is possible for the Convolutional Neural Network to provide false positives when detecting spills by learning the features of the tiles and not the spilled liquids. A breakdown of the training and testing dataset can be seen below in table 9.

| Train | | Test | |
|---|---|---|---|
| Spills on Tiles | | Spills on Tiles | |
| Milk | 940 | Milk | 80 |
| Orange | 860 | Orange | 60 |
| Red | 980 | Red | 70 |
| Coffee/Tea | 770 | Coffee/Tea | 60 |
| Water | 800 | Water | 90 |
| Total | 4,350 | Total | 360 |
| | | | |
| Tiles with no spills | | Tiles with no spills | |
| White | 130 | White | 30 |
| Brown | 210 | Brown | 40 |
| cream | 160 | cream | 30 |
| Total | 600 | Total | 100 |

TABLE 10: DATASET USED FOR SPILLS ON TILES

### 4.3.2 Training method

The performance of the proposed Convolutional Neural Network models will depend on a number of factors that can be controlled during training. The following aspects were considered during the training of the Convolutional Neural Network models, The tuning of the training parameters, data augmentation, batch size, early stopping and mini-batch gradient descent.

Tuneable parameters are a method of improving the performance of a convolutional neural network is to alter its training parameters such as epochs or maximum iterations, learning rate, and loss functions. It is considered a common problem for Convolutional Neural Network to over train and become subjected to overfitting. Overfitting is an issue where the Convolutional Neural Network over trains and fits too well to the training dataset and becomes difficult for the model to generalise to the test dataset or new examples when introduced. This can be mitigated by limiting the epochs a model can train for and stopping training when the model starts to lose accuracy. As Convolutional Neural Networks are trained predominantly using a stochastic gradient descent algorithm the learning rate can be altered to change the amount a weight is changed according to the error rate during back propagation. It is crucial for the correct learning weight to be selected as models can converge too rapidly to a suboptimal solution or the learning rate can be too small and cause the training process to halt. The loss function of a convolutional network is a division of its optimization algorithm and is responsible for estimating the losses of a model so that the weights can be updated to reduce loss during the next epoch.

Image Data augmentation is a technique that is commonly used in convolutional neural network training as a means of artificially expanding the size of training dataset by creating altered versions of images within the training dataset. Images involved in data augmentation may have their brightness varied, colours altered, rotational plan shifted, and image reflected. This method can increase the training dataset by a significant amount and can result in models increased performance when generalising to new data.

### 4.3.3 Evaluation metrics

There is a limited number of methods that can be used to evaluate the performance of a convolutional neural network, with the primary metric being the accuracy, precision, recall and F1 score. The equations for each of these metrics can be seen in below in equation 8 to 11.

$$Precision = \frac{TP}{TP+FP} \quad \textbf{(8)}$$

$$Recall = \frac{TP}{TP+FN} \quad \textbf{(9)}$$

$$F1\ score = 2 * \frac{Precision*Recall}{Presion+Recall} \quad \textbf{(10)}$$

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \quad \textbf{(11)}$$

The True Positive (TP) and True Negative (TN) metrics refers to the number of objects that were correctly identified and correctly negatively identified, while The False Negative (FN) refers to the objects that were not detected by the model. While the metric False Positive (FP), refers to the number of objects that were falsely identified. Therefore, based on equation 8 the precision metric is the ratio of objects that have been positively identified compared to the rate of falsely identified objects. The recall metric measures the ratio of correctly detected objects compared to the baseline or ground truth in the training dataset and is defined in equation 9. The accuracy and F1 score are commonly used metrics to evaluate the performance of convolutional neural networks. While the accuracy metric is a widely used metric which can be used to summarize the performance of the model, which takes into consideration the true positive and negative metrics, while the F1 score takes into consideration the False positive and negative metrics. The F1 score is considered to be an overall indicator to performance of the model in relation to the dataset being used. In the research conducted, both the accuracy and F1

score were calculated using equation 10 and 11 respectively. The final results are presented below in table 11 and 12.

## 4.4 Results and Discussion

Four different architectures were explored as a means to determine the best convolutional neural network for the object detection of liquid spills on tiles. The liquid spills on tiles dataset used for this research contained 5,410 images with 4,710 images used for training and 700 images used for testing. The images taken at a resolution of 1920x1080 pixels and scaled down to 1280x720 pixels. Each image within the dataset was labelled using the open-source labeling software LabelImg and subsequently provided the data for the input layer of the Convolutional Neural Network models. The model was trained using TensorFlow and the Keras library as well an using a custom training script to improve the overall accuracy of each model. The standard TensorFlow model fit function was used to evaluate each model and provided the training accuracy, validation accuracy and test accuracy. The training took place on a NVidia 3080 graphics card that featured 10GB video ram. The training and testing results are presenting in Table 11 and 12.

| Training | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 4507 | 416 | 27 | 0.9940 | 0.9155 | 0.9532 | 0.9105 |
| MobileNet_V2 | 4326 | 560 | 64 | 0.9854 | 0.8854 | 0.9327 | 0.8739 |
| VGG-16 | 1918 | 2456 | 576 | 0.7690 | 0.4385 | 0.5585 | 0.3875 |
| Inception_V3 | 4762 | 172 | 16 | 0.9967 | 0.9651 | 0.9806 | 0.9620 |

TABLE 11: TRAINING RESULTS OF TILES DATASET

| Test | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 408 | 46 | 6 | 0.9855 | 0.8987 | 0.9401 | 0.8870 |
| MobileNet_V2 | 143 | 239 | 78 | 0.6471 | 0.3743 | 0.4743 | 0.3109 |
| VGG-16 | 162 | 256 | 42 | 0.7941 | 0.3876 | 0.5209 | 0.3522 |
| Inception_V3 | 347 | 90 | 23 | 0.9378 | 0.7941 | 0.8600 | 0.7543 |

TABLE 12: TESTING RESULTS OF TILES DATASET

It can be seen in Table 11 above that Inception_V3 performed the highest out of the selected architectures during the training phase, with a testing accuracy of 96%. However, MobileNet_V1 scored the highest during the testing phase, with an accuracy of 88%. It can be seen that MobileNet_V1 did not experience much variation between its testing and training results, with only a slight deviation of 3% from the testing accuracy of 91%. It can be seen that MobileNet_V2 and Inception_V3 have experienced overfitting, as their results have a variation of 56% and 21%, respectively. Overfitting occurs in a Convolutional Neural Network when the model over-tunes its parameters to the training dataset and starts to recognise specific images instead of individual features within the image. When this occurs, it becomes difficult for the model to generalise what features it looks for when presented with a new image outside the training set. This is more common when using more extensive networks as the advanced architectures allow for more information to be stored within the network and can allow for whole images to be recognised. It can be seen that VGG-16 performed poorly on both the training and testing dataset, scoring only 38% and 35%, respectively. The progress of training per epoch is reported in Figure 25.

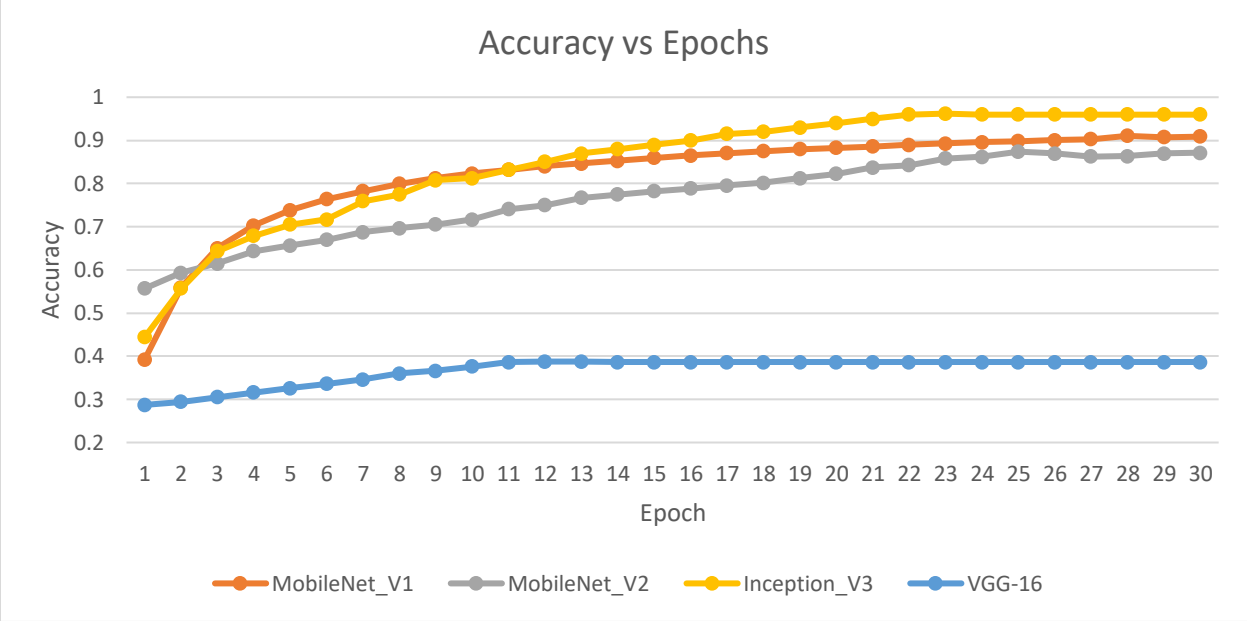



FIGURE 25: TRAINING ON TILES DATASET

The false negatives of each model are considerable high and concerning as the false positive rate presents spills that have not been detected and can lead to the fall of an individual. For example, MobileNet_V1 did not correctly identify a spill in 648 images out of 4,360 images. This may result in potential unidentified spills in real-life situations. The false-positive rates can be reduced by increasing the size of the training dataset, along with increasing or decreasing the size of the model's architecture.

## 4.5 QUANTISATION

The training of convolutional neural networks is an increasingly resource intensive process and not only incur significant computational cost, but also consume vast amounts of memory. Quantisation is required for machine learning models to be ran on devices such as the Google Coral and Raspberry Pi. The fundamental idea behind quantization is the conversion of weights and inputs to integer-based data. Traditionally machine learning models, especially convolutional neural networks are based on each weight being a 64-bit floating-point data type. Quantisation converts these 64-bit floating-point data types to 8-bit integers. This allows for the architecture to consume less memory and reduce the computational cost of the network, resulting in faster training times, at the expense of the accuracy of the model. Quantisation essentially reduces the model's complexity and computational cost to a quarter of its original cost. Quantisation also allows for models to be able to operate on computational restricted devices, such as smart phones or embedded system platforms. The results of the quantised models are presented in Table 13 and 14.

| Training | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 4268 | 648 | 34 | 0.9921 | 0.8682 | 0.9260 | 0.8622 |
| MobileNet_V2 | 3684 | 1185 | 81 | 0.9785 | 0.7566 | 0.8534 | 0.7442 |
| VGG-16 | 1749 | 2565 | 636 | 0.7333 | 0.4054 | 0.5222 | 0.3533 |
| Inception_V3 | 4018 | 911 | 21 | 0.9948 | 0.8152 | 0.8961 | 0.8117 |

TABLE 13: QUANTISED TRAINING RESULTS ON TILES DATASET

| Test | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 394 | 57 | 9 | 0.9777 | 0.8736 | 0.9227 | 0.8565 |
| MobileNet_V2 | 121 | 255 | 84 | 0.5902 | 0.3218 | 0.4165 | 0.2630 |
| VGG-16 | 67 | 330 | 63 | 0.5154 | 0.1688 | 0.2543 | 0.1457 |
| Inception_V3 | 314 | 116 | 30 | 0.9128 | 0.7302 | 0.8114 | 0.6826 |

TABLE 14: QUANTISED TESTING RESULTS ON TILES DATASET

It can be seen in Tables 13 and 14 that the quantised training and testing results are similar to that of the previous section, with a slight decrease in accuracy. In the models tested, each model experienced an average of 10%-15% decrease in performance during testing and training. This decrease in performance was expected as the weights and biases were changed from a 64-bit floating point data type to an 8-bit integer data type. This results in a reduction in the precision of the model. It was seen that Mobilenet_V1's testing score decreased by 3% when comparing the original model and the quantised model. It can still be seen that MobileNet_V2 and Inception_V3 are suffering from overfitting during the quantised process. It can be noted that the VGG-16 performed worse during quantisation when compared to the original model's accuracies. The original model only saw a variation of 3%, while it saw a 16% variation while undergoing training when quantised.

## 4.6 Summary and Conclusion

This chapter presented the effective methodology for the use of MobileNet_V1, Mobiltnet_V2, VGG-16 and inception_V3 in liquid spills on tiles object detection. The detailed experiment and analysis of results have demonstrated the importance of data augmentation, and fine-tuning parameters, such as learning rate and batch size. The results within this chapter demonstrate the suitability of the proposed architectures for liquid spills in a common household environment and the protentional viability of creating a fall prevention device for the elderly community. The highest scoring model was the MobileNet_V1 in both the original and quantised tests resulting in an accuracy of 88% and 85%. It can be seen that during quantisation that the accuracy of the model drops by an insignificant amount and is considered to be a viable option for the detection of spills on hard surfaces. The false positives rate in both the standard and quantised models are considerable high and concerning as the false positive rate presents spills that have not been detected and can lead to the fall of an individual.

# CHAPTER 5: SPILLS DETECTION ON CARPET USING CONVOLUTIONAL NEURAL NETWORKS

## ABSTRACT

This chapter investigates whether detecting liquid spills on carpeted surfaces is possible using a Convolutional Neural Network. Liquid spills on carpeted surfaces are considered more difficult than on tiled surfaces, as the liquid is absorbed into the carpet. Therefore, it was investigated whether a Convolutional Neural Network would be able to detect spills on carpeted surfaces with a high degree of accuracy. To achieve this high degree of accuracy, multiple convolutional neural network models will be tested, such as MobileNet_V1, MobileNet_V2, Inception_V3 and VGG-16. The listed models will be trained on a dataset containing 6,730 images of spills that can be broken down into eight sets of classes, with four classes used for testing and four classes used as a control group. The carpet samples were classified as dark blue, dark grey, dark brown and light brown/cream. The training data consists of 6,370 images, with 3,740 images containing spills and 2,530 images containing no spills used as a control group.

The models will also be tested on a dataset of 460 images categorised into the eight classes listed above. The models will also be quantised and evaluated as quantising a model will reduce the accuracy of each model. The quantised models will be assessed on the Google Coral Development board to ensure that a developed model can operate on an embedded systems platform when in a simulated environment. The models under evaluation will be examined using the metrics of analysis, precision, recall, F1 score and accuracy. These metrics allow for comprehensive observation of the different architectures performed under the same training and testing environment. The MobileNet_V1 model outperforms the newer MobileNet_V2, VGG-16 and Inception_V3. However, it can be seen that the other models suffer from overfitting as a result of their complex architecture.

## 5.1 INTRODUCTION

Falls within the elder community can be devastating to an individual and result in a loss of confidence in walking, also known as a post-fall syndrome. The fear of falling can restrict an elderly individual's confidence in performing day-to-day activities, resulting in a decline in their quality of life. The World Health Organisation (WHO) estimates that 684,000 deaths occur by individuals falling annually, with 37.7 million falls occurring worldwide. The concern of this number increasing is dramatized throughout the study, along with the health care costs associated with each fall. Due to the nature of the data collection within the elderly community, there is a lack of data that investigates the surfaces on which falls occur. It is believed to be uncommon for an individual to experience a fall on a carpeted surface, as they are considered to absorb most liquids, unlike hard surfaces. However, it is still possible for these falls to occur in most age care settings and homes. Areas of homes such as bedrooms and lounge rooms typically have food consumed in them that can lead to spills.

This chapter aims to determine whether it is possible to detect liquid spills on carpeted flooring with the use of a non-linear machine learning model such as a Convolutional Neural Network. To achieve this high degree of accuracy, multiple convolutional neural network models will be tested, such as MobileNet_V1, MobileNet_V2, Inception_V3 and VGG-16. The listed models will be trained on a dataset containing 6,730 images of spills that can be broken down into eight sets of classes, with four classes used for testing and four classes used as a control group. The split liquids have been categorised as the carpet colour due to the liquid being absorbed into the carpet and losing its colour property. The carpet samples were classified as dark blue, dark grey, dark brown and light brown/cream. The training data consists of 6,730 images, with 3,740 images containing spills and 2,530 images containing no spills used as a control group. The training data also consists of a control group with four categories. This control group consists of the carpet samples with no spilt liquids on them and can be categorised as dark blue, dark grey, dark brown and light brown/cream.

## 5.2 METHODS

### 5.2.1 TRAINING DATASET

The training and testing dataset used for this research section consisted of 6,730 images with dimensions of 1280x720 pixels. The images were taken at various distances from the targeted spill, ranging from 0.30 metres to 4 meters. The images were also taken from multiple angles on both the vertical and horizontal axis, typically moving around the spill in 60-degree increments. The images were obtained using a GoPro Hero 9 and were taken at various times throughout the day to ensure multiple lighting conditions.

The dataset used to train the Convolutional Neural Networks contains eight sets of classes. The spilled liquids have been categorised into four categories based on the colour of the carpet used. Nine carpet samples were selected during this study and were classified together as the lighting within the environment made it difficult for the GoPro Hero 9 to distinguish between similar shades. The categories chosen for this study were dark blue, dark grey, dark brown and light brown/cream.

The dataset selected also consists of a subset of classes that will be used as a control class to ensure that the Convolutional Neural Network is not learning the features of the tiles when detecting the spill. These control classes consisted of the above categories, without liquids spilt on the carpet samples. The Convolutional Neural Network can provide false positives when detecting spills by learning the features of the tiles and not the spilled liquids. A breakdown of the training and testing dataset can be seen below in table 15.

| Spills on Carpet | | Spills on Carpet | |
|---|---|---|---|
| Cream | 680 | Cream | 60 |
| Light Grey | 1590 | Light Grey | 90 |
| Navy | 500 | Navy | 70 |
| Dark Grey | 970 | Dark Grey | 80 |
| Total | 3,740 | Total | 300 |
| | | | |

| Carpet no Spills | | Carpet no Spills | |
|---|---|---|---|
| Cream | 450 | Cream | 40 |
| Light Grey | 790 | Light Grey | 40 |
| Navy | 300 | Navy | 30 |
| Dark Grey | 990 | Dark Grey | 50 |
| Total | 2,530 | Total | 160 |

TABLE 15:DATASET USED FOR SPILLS ON TILES

The performance of the proposed Convolutional Neural Network models will depend on several factors that can be controlled during training. Therefore, the following aspects were considered during the training of the Convolutional Neural Network models, The tuning of the training parameters, data augmentation, batch size, early stopping and mini-batch gradient descent. A summary of the training parameters can be seen in section 4.3.2

## 5.3 Results and Discussion

During this research, four different architectures were explored to provide the best convolutional neural network for the object detection of liquid spills on tiles. The liquid spills on tiles dataset used for this research contained 6,730 images, with 6,270 images used for training and 460 images used for testing. Each image within the dataset was labelled using the open-source labelling software LabelImg and subsequently provided the data for the input layer of the convolutional neural network models. The model was trained using TensorFlow and the Keras library and a custom training script to improve the overall accuracy of each model. The standard TensorFlow model fit function was used to evaluate each model and provided the training, validation, and test accuracy. The movement took place on an NVidia 3080 graphics card that featured 10GB video ram. The training and testing results are presented in Table 16 and 17.

| Training | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 5872 | 357 | 41 | 0.9931 | 0.9427 | 0.9672 | 0.9365 |
| MobileNet_V2 | 5531 | 667 | 72 | 0.9871 | 0.8924 | 0.9374 | 0.8821 |
| VGG-16 | 2578 | 2998 | 694 | 0.7879 | 0.4623 | 0.5827 | 0.4112 |
| Inception_V3 | 5907 | 325 | 38 | 0.9936 | 0.9478 | 0.9702 | 0.9421 |

TABLE 16: TRAINING RESULTS ON CARPET DATASET

| Test | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 421 | 36 | 3 | 0.9929 | 0.9212 | 0.9557 | 0.9152 |
| MobileNet_V2 | 166 | 270 | 24 | 0.8737 | 0.3807 | 0.5304 | 0.3609 |
| VGG-16 | 147 | 262 | 51 | 0.7424 | 0.3594 | 0.4843 | 0.3196 |
| Inception_V3 | 361 | 92 | 7 | 0.9810 | 0.7969 | 0.8794 | 0.7848 |

TABLE 17: TESTING RESULTS ON CARPET DATASET

It can be seen in Table 16 and 17 above that Inception_V3 performed the highest out of the selected architectures during the training phase, with a testing accuracy of 94%. However, MobileNet_V1 scored the highest during the testing phase, with an accuracy of 91%. It can be seen that MobileNet_V1 did not experience much variation between its testing and training results, with only a slight deviation of 2% from the testing accuracy of 93%. It can be seen that MobileNet_V2 and Inception_V3 have experienced overfitting, as their results have a variation of 52% and 16%, respectively. It can be seen that VGG-16 performed poorly on both the training and testing dataset, scoring only 41% and 31%, respectively. It can be noted that each model did perform slightly better on the carpeted dataset in comparison to tiles dataset. It is believed that the overfitting seen by MobileNet_V2 and Inception_V3 is due to the complexity of their architecture. MobileNet_V2 is designed to handle a larger number of classes when compared to Mobilenet_V1. This can be demonstrated in the results when comparing both MobileNet_V1 and MobileNet_V2, as MobileNet V2's architecture may be too complex when there is a small number of classes involved. The Model may become prone to overfitting due to the complexity of the architecture on a small dataset. It can be seen that VGG-16 performed poorly on both the tiles and carpet dataset and is believed to be due to its lack of specialised features within the architecture. The architecture is considered basic when compared to MobileNet_V2 as it lacks features such as the depthwise separable convolutional, linear bottlenecks and shortcuts between the linear bottlenecks. The progress of training per epoch is reported in Figure 26.

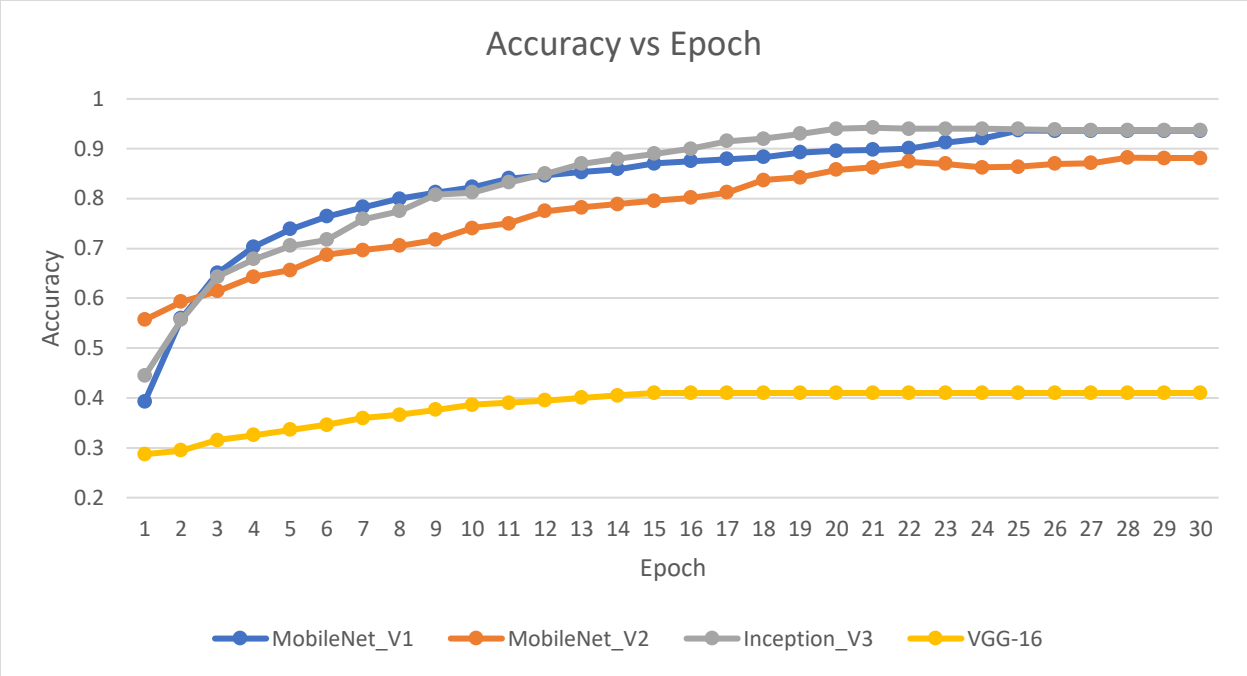



FIGURE 26: TRAINING ON CARPET DATASET

It can be seen that each model recorded a high number of false negatives, which can be considered to be concerning as these can lead to a potential fall as the system may incorrectly identify a spilt liquid. It can be seen that all models performed slightly better on the carpet dataset compared to the tile dataset in the previous chapter. This is thought to mainly be because most images within the carpet dataset experience a noticeable change when a spill occurs due to the liquid absorbed by the carpet. In contrast, liquids on tiled surfaces tend to reflect or become transparent as they spread out on the surface.

## 5.4 QUANTISATION

Convolutional Neural Networks have proven to be a powerful machine learning model for image classification. One drawback however is the high computational cost requirements to run the machine learning models. The computational costs of these models can be reduced reducing the weights and biases of the models. The fundamental idea behind quantization is the conversion of weights and inputs to integer-based data. Traditionally machine learning models, especially convolutional neural networks, are based on each weight being a 64-bit floating-point data type. Quantization converts these 64-bit floating-point data types to 8-bit integers. This allows for the architecture to consume less memory and reduce the computational cost of the network, resulting in faster training times at the expense of the model's accuracy. As a result, quantization reduces the model's complexity and computational cost to a quarter of its original price. Quantization also allows models to operate on restricted computational devices, such as smartphones or embedded system platforms. The results of the quantized models are presented in table 18 and 19.

| Training Quantised | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 5564 | 661 | 45 | 0.9920 | 0.8938 | 0.9403 | 0.8874 |
| MobileNet_V2 | 4787 | 1406 | 77 | 0.9842 | 0.7730 | 0.8659 | 0.7635 |
| VGG-16 | 1957 | 3594 | 719 | 0.7313 | 0.3525 | 0.4758 | 0.3121 |
| Inception_V3 | 4962 | 1255 | 53 | 0.9894 | 0.7981 | 0.8835 | 0.7914 |

TABLE 18: QUANTIZED TRAINING RESULTS ON CARPET DATASET

| Test Quantised | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 402 | 53 | 5 | 0.9877 | 0.8835 | 0.9327 | 0.8739 |
| MobileNet_V2 | 114 | 325 | 21 | 0.8444 | 0.2597 | 0.3972 | 0.2478 |
| VGG-16 | 74 | 329 | 57 | 0.5649 | 0.1836 | 0.2772 | 0.1609 |
| Inception_V3 | 302 | 152 | 6 | 0.9805 | 0.6652 | 0.7927 | 0.6565 |

TABLE 19: QUANTIZED TESTING RESULTS ON CARPET DATASET

It can be seen in Tables 18 and 19 that the quantised training and testing results are similar to that of the previous section, with a slight decrease in accuracy. In the models tested, each model experienced an average of 10%-15% decrease in performance during testing and training. This decrease in performance mirrors that of chapter 4. It was seen that Mobilenet_V1's experienced a variation of 1% when comparing the testing and training accuracy of 88% and 87%. It can be noted that MobileNet_V1 experienced the highest training accuracy in the quantised test out of the selected models. In contrast, Inception_V3 scored the highest in the non-quantised tests. It can still be seen that MobileNet_V2 and Inception_V3 are suffering from overfitting during the quantised process and experienced a variation of 52% and 14%, respectively. It can be noted that the VGG-16 performed worse during quantisation when compared to the original model's accuracies. It can be seen that each architecture has similar results to that of the previous chapter. MobileNet_V1 experience an increase of accuracy of 2% for both the testing and training when comparing the tiles and carpeted dataset. The original model only saw a variation of 10%, while it saw a 15% variation while undergoing training when quantised.

## 5.5 Summary and Conclusion

This chapter presented the effective methodology for the use of MobileNet_V1, Mobiltnet_V2, VGG-16 and inception_V3 in liquid spills on carpet object detection. The detailed experiment and analysis of results have demonstrated the importance of data augmentation, and fine-tuning parameters, such as learning rate and batch size. The results within this chapter demonstrate the suitability of the proposed architectures for liquid spills in a common household environment and the protentional viability of creating a fall prevention device for the elderly community. The highest scoring model was the MobileNet_V1 in both the original and quantised tests resulting in an accuracy of 88% and 85%. It can be seen that during quantisation that the accuracy of the model drops by an insignificant amount and is considered to be a viable option for the detection of spills on hard surfaces.

# CHAPTER 6: SPILLS DETECTION ON TILES AND CARPET USING CONVOLUTIONAL NEURAL NETWORKS

## ABSTRACT

The research noted below governs whether it is possible to detect liquid spills on carpeted and tiled surfaces with the use of a non-linear machine learning model such as a Convolutional Neural Network. It was hypothesised that a Convolutional Neural Network will be able to distinguish spills on carpeted and tiled surfaces to a high degree of accuracy. To attain this degree of accuracy, numerous convolutional neural network models will be compared, including MobileNet_V1, MobileNet_V2, Inception_V3 and VGG-16. The highest scoring model will be selected and used in the simulated motion test in chapter 7. These models will be trained on the previous datasets from chapter four and five and will containing 11,200 images of spills. These again will be divided into 16 sets of classes categorised into the following: Milk, Orange, Red, Coffee/Tea and water for tiled surfaces and cream, light grey, navy and dark grey for the carpeted surfaces. The training data also contains 7 classes of tiles and carpets including no spills as a control group in this research. The classes with no spills consist of Brown, Light and White for tiles and cream, light grey, navy and dark grey for the carpeted surfaces. The testing dataset of 920 images that are categorised into the 8 classes that are noted above will also be used to test the model. The models will then be reduced by quantising them and evaluated on the Google Coral Development board. This will guarantee that a developed model will operate on an embedded systems platform when in a simulated environment. The models that are under evaluation will be examined using the aforementioned metrics of analysis, recall, precision, accuracy and F1 score. The research concluded that the MobileNet_V1 model outperformed the newer MobileNet_V2, Inception_V3 and VGG-16. The other models suffered from overfitting due to their complex architecture.

## 6.1 INTRODUCTION

Health Direct Victoria [5] has indicated that elderly Australians have a high frequency of serious falls, with 2 in 3 occurring within a household environment. This can often be due to spills, poor lighting, and other environmental factors. In addition to this, it is noted that 1 in 3 elderly Australians will experience a severe fall every 12 months, with 1 in 5 of these falls resulting in the person requiring support from a hospital. The research also indicates that falls are more likely to occur on hard surfaces, including laminated flooring and tiles, as these materials have the ability to become exceedingly slippery when wet and fail to absorb liquid. As it is difficult for selected Convolutional Neural Networks to detect certain liquids, it is crucial to train them to detect spills on both tiled and carpeted surfaces. This will provide the models with more data to help correctly identify spills. For example, if the network experiences a water spill on a tiled surface, it may not correctly identify it due to the lighting in the vicinity. However, if the spill happens to leak from the tiles onto a carpeted surface, then the model will be able to identify the spill correctly.

The current research aims to explore the possibility of detecting liquid spills on laminated and tiled flooring with the use of non-linear machine learning models including Convolutional Neural Networks. Numerous versions of these networks will be utilised to increase accuracy including MobileNet_V1, Inception_V3, MobileNet_V2, and VGG-16. These will be trained on a dataset containing 11,200 images of spills segmented into 16 sets of classes. The data contains 3 classes of tiles containing no spills as a control group for the experiment, which this no spills subclass consisting of White, Brown, and Light. The dataset will also contain four classes of carpets within the no spill subclass. It is also important to note that the light category contains a variety of cream and light brown coloured tiles. In addition to this, the spilled liquids have been categorised into 9 classes: Milk, Orange, Red, Coffee/Tea and Water for tiled surfaces and cream, light grey, navy and dark grey for the carpeted surfaces.

## 6.3 METHODS

### 6.3.1 TRAINING DATASET

The training and testing dataset used for this section of the research consisted of 12,120 images with dimensions of 1280x720 pixels. The images were taken at various distances from the targeted spill, ranging from 0.30 metres to 4 meters. The images were also taken from various angles on both the vertical and horizontal axis, typically moving around the spill in 60-degree increments. The images were obtained by using a go pro hero 9 and were taken at various times throughout the day to ensure a variety of lighting conditions.

The dataset used to train the Convolutional Neural Networks contains 16 sets of classes. The spilled liquids have been categorised into 9 classes: Milk, Orange, Red, Coffee/Tea and Water for tiled surfaces and cream, light grey, navy and dark grey for the carpeted surfaces. Each one of these classes contains various spilled drinks but has been grouped together as the camera used within this experiment could not distinguish the difference between some types of liquids. The liquids that were grouped together consisted of drinks such as black currant juice and raspberry soft drink under the red spill class and tea and coffee under the same class. This was due to the quality of the camera and the lighting within the test environment providing a dull lighting environment, along with the colour of the flooring underneath the spilt liquid making it difficult to distinguish colours clearly.

The dataset selected also consists of a subset of classes that are going to be used as a control class to ensure that the Convolutional Neural Network is not learning the features of the tiles when detecting the spill. These control classes consisted of white, brown, and cream tiles. It is possible for the Convolutional Neural Network to provide false positives when detecting spills by learning the features of the tiles and not the spilled liquids. A breakdown of the training and testing dataset can be seen below in table 9.

| Training | | Testing | |
|---|---|---|---|
| Images Total | 11,200 | Images Total | 920 |
| | | | |
| Spills on Tiles | | Spills on Tiles | |
| Milk | 940 | Milk | 80 |

| | | | |
|---|---|---|---|
| Orange | 860 | Orange | 60 |
| Red | 980 | Red | 70 |
| Coffee/Tea | 770 | Coffee/Tea | 60 |
| Water | 800 | Water | 90 |
| Total | 4,350 | Total | 360 |
| | | | |
| Tiles with no spills | | Tiles with no spills | |
| White | 130 | White | 30 |
| Brown | 210 | Brown | 40 |
| Light | 160 | Light | 30 |
| Total | 600 | Total | 100 |
| | | | |
| Spills on Carpet | | Spills on Carpet | |
| Cream | 680 | Cream | 60 |
| Light Grey | 1590 | Light Grey | 90 |
| Navy | 500 | Navy | 70 |
| Dark Grey | 970 | Dark Grey | 80 |
| Total | 3,740 | Total | 300 |
| | | | |
| Carpet no Spills | | Carpet no Spills | |
| Cream | 450 | Cream | 40 |
| Light Grey | 790 | Light Grey | 40 |
| Navy | 300 | Navy | 30 |
| Dark Grey | 990 | Dark Grey | 50 |
| Total | 2,530 | Total | 160 |

TABLE 20: DATASET USED FOR SPILLS ON TILES

The performance of the proposed Convolutional Neural Network models will depend on several factors that can be controlled during training. Therefore, the following aspects were considered during the training of the Convolutional Neural Network models, The tuning of the training parameters, data augmentation, batch size, early stopping and mini-batch gradient descent. A summary of the training parameters can be seen in section 4.3.2

## 6.3 Results and Discussion

During this research four different architectures were explored as a means to provide the best convolutional neural network for the object detection of liquid spills on tiles. The liquid spills on tiles dataset used for this research contained 12,120 images with 11,200 images used for training and 920 images used for testing. Each image within the dataset was labelled using the open-source labeling software LabelImg and subsequently provided the data for the input layer of the convolutional neural network models. The model was trained using TensorFlow and the Keras library as well an using a custom training script to improve the overall accuracy of each model. The standard TensorFlow model fit function was used to evaluate each model and provided the training accuracy, validation accuracy and test accuracy. The training took place on a NVidia 3080 graphics card that featured 10GB video ram. The training and testing results are presenting in Table 21 and 22.

| Training | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 10438 | 705 | 57 | 0.9946 | 0.9367 | 0.9648 | 0.9320 |
| MobileNet_V2 | 10214 | 879 | 107 | 0.9896 | 0.9208 | 0.9540 | 0.9120 |
| VGG-16 | 4531 | 5820 | 849 | 0.8422 | 0.4377 | 0.5761 | 0.4046 |
| Inception_V3 | 10683 | 435 | 82 | 0.9924 | 0.9609 | 0.9764 | 0.9538 |

TABLE 21: TRAINING RESULTS ON TILED AND CARPETS DATASET

| Test | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 852 | 27 | 41 | 0.9541 | 0.9693 | 0.9616 | 0.9261 |
| MobileNet_V2 | 356 | 466 | 98 | 0.7841 | 0.4331 | 0.5580 | 0.3870 |
| VGG-16 | 269 | 475 | 176 | 0.6045 | 0.3616 | 0.4525 | 0.2924 |
| Inception_V3 | 741 | 95 | 84 | 0.8982 | 0.8864 | 0.8922 | 0.8054 |

TABLE 22: TEST RESULTS ON TILED AND CARPET DATASET

It can be seen in Tables 21 and 22 above that the MobileNet_V1 model outperforms the MobileNet_V2, VGG-16 and Inception_V3 by a substantial amount in testing. MobileNet_V1 achieved 12% higher than Inception_V3. MobileNet_V1 and Inception_V3 scored 92% and 80% respectively. While only experiencing a variation of 1% and 15%, respectively. It can be seen that VGG-16 scored higher on this dataset when compared to previous chapters and saw a deviation of 11%, with a training and testing accuracy of 40% and 29%, respectively. While MobileNet_V2 experienced overfitting and saw a variation of 53%, similar to chapters four and five. It can be noted that each model's performance increased during the training and testing of this dataset when compared to the datasets used in chapters four and five. It is believed this is due to the increase of images available to the models during training. It is believed that the increase of images helped to reduce the amount of overfitting experienced by MobileNet_V2 and Inception_V3. The progress of training per epoch is reported in Figure 27.

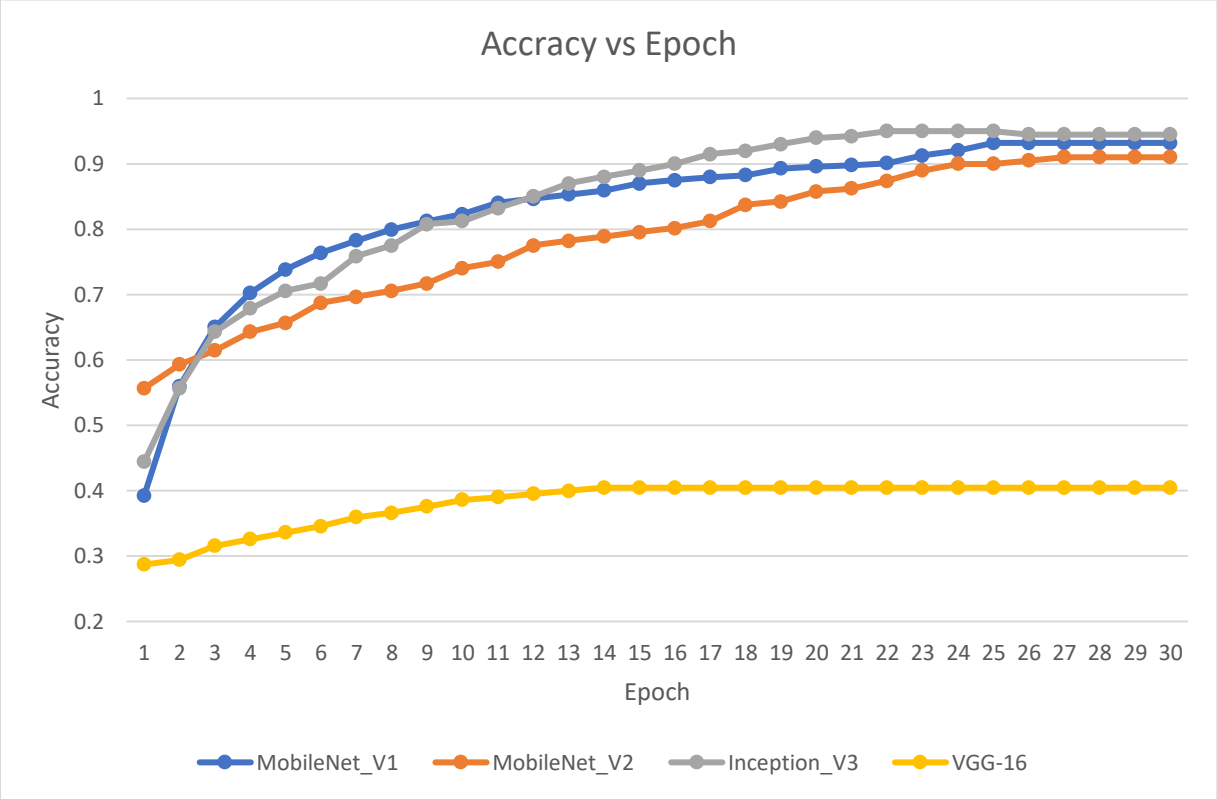



FIGURE 27: TRAINING ON TILES AND CARPET DATASET

## 6.4 QUANTIZATION

Convolutional Neural Networks are a very resource intensive algorithm. They do not only require a significant computational cost, but they also consume a large amount of memory when operating. The fundamental idea behind quantization is the conversion of weights and biases to integer type. This can result in a large reduction in memory usage and a reduction in computational power required for each inference of the model. Quantization is ideal when looking to run machine learning tasks on an embedded system devices as they have limited memory and computational power when compared to a traditional computer. The results of the quantised models are presented in Table 23 and 24.

| Training | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 10198 | 940 | 62 | 0.9940 | 0.9156 | 0.9532 | 0.9105 |
| MobileNet_V2 | 10131 | 921 | 148 | 0.9856 | 0.9167 | 0.9499 | 0.9046 |
| VGG-16 | 3962 | 6346 | 892 | 0.8162 | 0.3844 | 0.5226 | 0.3538 |
| Inception_V3 | 10784 | 320 | 96 | 0.9912 | 0.9712 | 0.9811 | 0.9629 |

TABLE 23: QUANTIZED TRAINING RESULTS ON TILES AND CARPETS DATASET

| Test | True Positive (TP) | False Negative (FN) | False Positive (FP) | Precision | Recall | F1 score | Accuracy |
|---|---|---|---|---|---|---|---|
| MobileNet_V1 | 823 | 55 | 42 | 0.9514 | 0.9374 | 0.9443 | 0.8946 |
| MobileNet_V2 | 350 | 476 | 94 | 0.7883 | 0.4237 | 0.5512 | 0.3804 |
| VGG-16 | 302 | 435 | 183 | 0.6227 | 0.4098 | 0.4943 | 0.3283 |
| Inception_V3 | 722 | 107 | 91 | 0.8881 | 0.8709 | 0.8794 | 0.7848 |

TABLE 24: QUANTIZED TESTING RESULTS ON TILES AND CARPET DATASET

It can be seen in Tables 23 and 24 that the quantised training and testing results are similar to that of the previous section, with a slight decrease in accuracy. It can be seen that MobileNet_V1 scored the highest out of the selected architectures. The MobileNet_V1 architecture performed higher on the tiles and carpet dataset than the tiles and carpet datasets alone, by 4% and 2%, respectively. VGG-16 experienced an improvement in its testing accuracy when compared to the previous quantised results in chapters four and five. However, it can be

seen that VGG-16 suffered a variation of 2%, while in chapter four experienced a variation of 16%. MobileNet_V2 experienced similar results to chapters four and five with a slight improvement in its accuracy. It can still be noted that with a larger dataset, MobiltNet_V2 still suffered from overfitting. It can be noted that Inception_V3 showed slight improvement when compared to previous chapters and saw a variation of 18% between the training and testing datasets.

## 6.5 Summary and Conclusion

This chapter presented the effective methodology for the use of MobileNet_V1, Mobiltnet_V2, VGG-16 and inception_V3 in liquid spills on tiles object detection. The detailed experiment and analysis of results have demonstrated the importance of data augmentation, and fine-tuning parameters, such as learning rate and batch size. The results within this chapter demonstrate the suitability of the proposed architectures for liquid spills in a common household environment and the protentional viability of creating a fall prevention device for the elderly community. The highest scoring model was the MobileNet_V1 in both the original and quantised tests resulting in an accuracy of 88% and 85%. It can be seen that during quantisation that the accuracy of the model drops by an insignificant amount and is considered to be a viable option for the detection of spills on hard surfaces.

# CHAPTER 7: EMBEDDED SYSTEMS APPLICATION

## ABSTRACT

This study determines whether it is possible for an embedded system to detect liquid spills on tiled and carpeted surfaces while attached to an individual's waist while moving. The embedded systems chosen for this experiment were the Google Coral Development Board, the Raspberry Pi 4B and the Raspberry Pi 4B with the addition of the Coral USB Accelerator. The embedded systems platform was placed on the individual's waist, as this is the centre of mass for most individuals and reduced unwanted motion while traversing. It was hypothesised that a convolutional network would be able to detect spills on tiled surfaces to a high degree of accuracy, with the Google Coral Development board providing the best performance. The MobileNet_V1 architecture was chosen for this experiment as it provided the highest accuracy compared to the previous models evaluated. MobileNet_V1 was trained on the previous datasets from chapter six and will contain 11,200 images of spills. These again will be divided into 16 sets of classes categorised into the following: Milk, Orange, Red, Coffee/Tea and water for tiled surfaces and cream, light grey, navy and dark grey for the carpeted surfaces. The training data also contains seven classes of tiles and carpets, including no spills as a control group in this research. The classes with no spills consist of Brown, Light and White for tiles and cream, light grey, navy and dark grey for the carpeted surfaces. The testing dataset of 920 images that are categorised into the eight classes that are noted above will also be used to test the model. The embedded systems were placed on the individual's waist and tested in a simulated dynamic motion test, where the individual approached a spill on the ground.

## 7.1 INTRODUCTION

Falls within the elder community can be devastating to an individual and result in a loss of confidence in walking, also known as a post-fall syndrome. The fear of falling can restrict an elderly individual's confidence in performing day-to-day activities, resulting in a decline in their quality of life. There is currently limited research on mobile aid devices that use artificial intelligence models to assist in fall prevention in the elderly community. Many research approaches focus on the care provided to an individual after a fall has taken place, without concern for preventing the initial fall. It has been shown throughout publications that there is a need for a mobile aid device that can aid in preventing falls within the elderly community. With the recent development of low power devices such as the Google Coral Development board, it is possible to run machine learning models at high inferencing speeds allowing for real time object detection.

The current research aims to explore the possibility of using both the Google Coral Development board and the Raspberry Pi 4B in a simulated motion test. The chosen architecture for this test includes MobileNet_V1 as this architecture was the highest scoring on the testing dataset in previous section. The MobileNet_V1 architecture was trained on a dataset containing 11,200 images of spills segmented into 16 sets of classes. During previous testing datasets MobileNet_V1 typically scored above 90% accuracy, making it the ideal candidate to test in the simulated motion test. The power consumption of each device was also tested during this experiment.

## 7.2 METHOD

Once the training of the MobileNet_V1 model is completed, it is required to be quantised and transferred to the Google Coral development board and Raspberry Pi. This process essentially converts the original TensorFlow model to a TensorFlow Lite model, which is required to run on embedded systems and single-board computers. The MobileNet_V1 model can be transferred to the raspberry pi via a USB, whereas transferring the TensorFlow Lite models across to the Google Coral Board along with the Python script used to run the models requires the use of the Mendel Development Tool. The Mendel Development Tool is a command-line tool that allows for interfacing between the Mendel operating system on the Google Coral Development Board and a regular computer. Files can be pushed between the computer and Google Coral development Board with the use of the push and pull command. To run the TensorFlow Lite model, a custom Python script needs to be written that defines the source video input and location of the model, along with other features such as bounding boxes and recording the accuracy within the model. The source input of the model is provided by the use of the Google Coral Camera for the Google Coral Development Board and the Raspberry Pi Camera for the Raspberry Pi 4B. The Google Coral Accelerator can be connected to the Raspberry Pi 4B via the onboard USB port. Due to the architecture being converted to a TensorFlow Lite model, the only requirements are for a slight alteration in the Python script to run the model.

A housing unit was created for both the Google Coral Development Board and the Raspberry Pi 4B as a means of attaching them to an individual's waist. The housings featured a small cut-out on the rear, which allowed for a belt to pass through. The systems were then attached to the individual's waist using the belt loops typically found on jeans. The embedded systems platform was placed on the individual's waist, as this is the centre of mass for most individuals and reduced unwanted motion while traversing. The Google Coral Development board used the Google Coral Camera as a video input for the system, while the Raspberry Pi 4B used the Raspberry Pi Camera. Both of these cameras were set to a resolution of 1280x720p. During this test, the devices were powered using a USB-C power bank of 4,000 mAh and the power consumption was recorded using a Eversame USB Power Meter. The Power Meter was connected in line with the USB-C power cable supplied by the USB-C power bank. The power

meter was reset after the model was loaded and running and the average power consumption in mAh was recorded.

To conduct the test, a spill was placed on a tiled or carpeted surface 2 metres away from the individual's starting location. The individual was then asked to move forward towards the spills at an average walking rate. This was repeated multiple times until sufficient evidence was collected.

## 7.3 Results and Discussion

During this research, the MobileNet_V1 was selected to be trained and quantised on the tiles and carpet dataset, which contained 11,200 images. The models were then transferred to the Google Coral Development board and the Raspberry Pi 4B with and without the aid of the Google Coral USB Accelerator. A custom python script was written to run the model, draw the bounding boxes, and record each device's frame rate. Each device was then attached to an individual at waist level, with the individual moving towards a spill at an average walking pace. The power consumption was recorded using an Eversame Power Meter Tester. The average frame rate, inferencing time and power consumption are reported below in Table 25 to 27.

| | Google Coral Development Board | Raspberry Pi 4B | Raspberry Pi 4B + Accelerator |
|---|---|---|---|
| Average Frame Rate (FPS) | 174.82 | 4.21 | 16.98 |

TABLE 25: AVERAGE FRAME RATE OF TESTED DEVICES

| | Google Coral Development Board | Raspberry Pi 4B | Raspberry Pi 4B + Accelerator |
|---|---|---|---|
| Inference time (ms) | 5.72 | 237.52 | 58.89 |

TABLE 26: AVERAGE INFERENCING TIME OF TESTED DEVICES

| | Google Coral Development Board | Raspberry Pi 4B | Raspberry Pi 4B + Accelerator |
|---|---|---|---|
| Power Consumption (mAh) | 734 | 648 | 813 |

TABLE 27: AVERAGE POWER CONSUMPTION OF TESTED DEVICES

Table 25 demonstrates the average frame rate of each device. The frame rate and interfacing speed are considered to be a similar metric and are interchangeable as they are both a direct representation of the systems performance. The Google Coral Development board had an average frame rate of 174.82 frames per second and outperformed the Raspberry Pi by 4350%. It is believed that this is due to the Google Coral Development board using a Tensor Processing Unit and the Raspberry Pi using a standard Central Processing Unit for the machine learning task. This can be considered an unfair comparison as the Raspberry Pi is not designed for machine learning tasks and is considered a single-board computer for a hobbyist. The frame rate of the Raspberry Pi is considered too low to perform real-time object detection. The Google Coral USB Accelerator did provide the Raspberry Pi with an improvement of 400% and had an average frame rate of 16 frames a second. The Google Coral Accelerator makes the Raspberry Pi a viable option for real-time object detection but may face challenges in fast-moving scenes of motion.

It can be noted in Table 26 that the power consumption of each board is relatively similar. It should be noted that the python script used on these devices was not optimised to reduce power consumption or to maximise performance. The Google Coral Development board was recorded to have an average power consumption of 734 mAh while running at 174 frames per second. It is believed that the power consumption can be lowered by limiting the frame rate at which the system was operating. The camera used for the Google Coral Development board had a frame limit of 60 frames per second, meaning that 114 frames were not required to be analysed by the model. This may lead to a significant reduction in the board's power consumption. The Google Coral Development Board was considered to be the most power-efficient as it used 4.21 mAh per frame, while the Raspberry Pi used 153.91 mAh per frame and the Raspberry Pi with the Google Coral Accelerator used 47.87 mAh per frame.

During the testing of the Google Coral development board and Raspberry Pi 4B, it can be noted that each board correctly identified the test spills. It can be noted that average accuracy of each model could not be accurately determined, as the accuracy changed depending on the position of the system. It was observed that each system had similar accuracies and varied between 10-30% accuracy when approximately 2.5 meters away. When approaching the spills, the systems accuracy increased to 50-80%. However, it can be noted that the devices did not identify spills nearing the end of passing over them. This was expected as the spills were nearly

out of frame, and sufficient data was not present to make a correct identification. Figures 28, 29 and 30 demonstrate the Google Coral Development board, Raspberry Pi 4B and Raspberry Pi 4B and Google Coral Accelerator.


**FIGURE 28: GOOGLE CORAL DEVELOPMENT BOARD SPILL TEST**


**FIGURE 29: RASPBERRY PI 4B SPILL TEST**


**FIGURE 30: RASPBERRY PI 4B WITH THE GOOGLE CORAL ACCELERATOR SPILL TEST**

## 7.4 SUMMARY AND CONCLUSION

This chapter presented the effective methodology for the application of MobileNet_V1 to the Google Coral Development Board and Raspberry Pi 4B. The detailed experiment and analysis of results have demonstrated and highlighted the importance of new technology and the increase of performance provided. The results within this chapter present the average performance of each device, along with their power consumption and average accuracy. The Google Coral development board was the highest performing in each aspect that was investigated. The Google coral development board averaged a frame rate of 174.8 frames per second, with an interfacing speed of 5.31ms. It can be seen that the Raspberry Pi experienced an average frame rate of 4.21 frames and 16.98 with the Google Coral Accelerator connected. The power consumptions of the Google Coral is slightly higher than the Raspberry Pi by 83 mAh. The Google Coral was determined to be a more power efficient device when comparing the power consumption to performance.

# CHAPTER 8: FUTURE WORK AND CONCLUSION

## 8.1 SUMMARY

The research explored within this thesis has significantly influenced the design and functionality of the current research proposal. As the life expectancy and fall rates of elderly individuals are expected to increase, the care and medical attention required by each individual are expected to increase and burden the health system drastically. The development of new technology has provided a vast array of possibilities for the embedded systems community to help combat this problem. These technologies will provide the needed improvements to make object detection viable on low-power portable devices. This will evidently allow for object recognition to be performed on systems that have a limited power supply, such as assisted walking devices, exoskeletons, and drones. In addition, these new technological advancements have allowed for the implementation of assisted walking devices in the aged care setting. These devices aim to aid in compensating for the decline of vision in the elderly and visually impaired, as poor vision is one of the leading factors of falls within these communities. Ultimately, it can be concluded that this research will be the foundation for a niche field of study and will help build upon other existing fields of study and assist in designing aid devices. The aims and findings of this work were carried out in four studies as follows.

1. Study one (Chapter Four) – To investigate the application of various machine learning models to spills on tiled surfaces.

In chapter four, the potential use of Convolutional Neural Networks to detect spilt liquids on tiled surfaces has been investigated and reviewed. The application of Convolutional Neural Networks to spills is a niche and has not been covered in previous research. It was demonstrated that MobileNet_V1 provided a high accuracy on the testing dataset when compared to MobileNet_V2, Inception_V3 and VGG-16. It was demonstrated throughout the chapter that Convolutional Neural Networks are a viable choice for real-time object detection of spilt liquids.

2. Study two (Chapter Five) – To investigate the application of various machine learning models to spills on carpet surfaces.

In Chapter Five, the potential use of Convolutional Neural Networks to detect spilt liquids on carpeted surfaces has been investigated and reviewed. It was demonstrated that MobileNet_V1 provided a high accuracy on the testing dataset when compared to MobileNet_V2, Inception_V3 and VGG-16. Therefore, it was determined that Convolutional Neural Networks are a viable machine learning option for the real-time detection of spills on carpeted surfaces.

3. Study three (Chapter Six) – To investigate the application of various machine learning models to spills on carpet surfaces.

In Chapter Six, the potential use of Convolutional Neural Networks to detect spilt liquids on tiled and carpeted surfaces has been investigated and reviewed. It was determined that Convolutional Neural Networks are a viable machine learning option for the real-time detection of spills on tiled and carpeted surfaces. Furthermore, it was demonstrated that MobileNet_V1 yielded the highest accuracy on the testing dataset when compared to MobileNet_V2, Inception_V3 and VGG-16.

4. Study four (Chapter Seven) – To investigate the application of Convolutional Neural Networks on various types of embedded systems.

Chapter Seven presented the practical methodology for the application of MobileNet_V1 to the Google Coral Development Board and Raspberry Pi 4B. The detailed experiment and analysis of results have demonstrated and highlighted the importance of new technology and the increase in performance provided. The results in this chapter present each device's average performance, power consumption, and accuracy. The Google Coral development board was the highest performing in each aspect that was investigated.

## 8.1 Limitations and future directions

In order to implement the findings of this thesis in real-life scenarios of fall prevention, there exists some limitations and challenges. The research conducted in this thesis does not robustly examine the device on participants and has no method of alerting individuals when a spill has been identified. The dataset used to train the Convolutional Neural Networks needs to be expanded to include more images of spills and potentially other objects that may cause a tripping or slipping hazard. The prototype can also be modified to reduce costs by removing some board components.

1. In order to apply the findings of this thesis to real-life fall prevention technology, some technical challenges exist. The datasets used within this research consist of a limited number of tiled and carpeted floorings. To allow for the real-life adaptation of this research, the dataset used will have to be expanded upon to include various types of carpets and tiles that were not included in this study. The limited data used within the research may have also led to overfitting for each model tested. Overfitting can lead to models performing poorly when shown a new object that were not included in the training data. In addition, the size of the dataset will need to be expanded upon to increase the accuracy and reduce the false negatives recorded during the research. Finally, additional objects, such as shoes and rugs, must be included in the dataset as these items commonly pose a tripping hazard for elderly individuals. These items were not included in this study, as large databases exist and have been widely utilised in this research field. Due to this, the research mainly focused on detecting liquids on various surfaces to aid in determining the viability of developing a device to aid in both fall and tripping prevention.

2. To achieve the ability to use the device in an aged care setting, rigorous prototype testing will need to be conducted to improve the device. This study lacked in-depth testing of the prototype on multiple individuals in various environments. The testing within this research focused on determining the correct machine learning architecture and embedded systems, along with determining the viability of detecting

liquids with Convolutional Neural Networks. Further research will need to be conducted to determine the viability of a fall-prevention device that follows the research.

3. The current prototype systems do not have the ability to alert users of potential slipping hazards. This issue can be solved with the use of audio feedback or haptic feedback. A study will need to be conducted to determine whether audio feedback, haptic feedback or both is more suited for elderly individuals

4. The development of a custom prototype will also need to be completed if this system is to be further explored. The current development boards selected host additional features that are not required under the current scope of the research. The development of a custom system will allow for decreased power consumption with a reduced cost and footprint.

# REFERENCES


[1] Mordor Intelligence, “WEARABLE TECHNOLOGY MARKET - GROWTH, TRENDS, COVID-19 IMPACT, AND FORECASTS (2021 - 2026),” 2018. [Online]. Available: https://www.mordorintelligence.com/industry-reports/wearable-technology-market. [Accessed 29 08 2021].

[2] Health Direct, “Falls and the elderly,” July 2020. [Online]. Available: https://www.healthdirect.gov.au/falls. [Accessed 05 June 2021].

[3] Australia Commission on Safety and Quality in Health Care, “Preventing Falls and Harm from Falls in Older People,” August 2009. [Online]. Available: https://www.safetyandquality.gov.au/sites/default/files/migrated/30458-Guidelines-RACF.pdf. [Accessed 3 June 2021].

[4] B. AK, Y. PD, W. SL and L. T, “Telerehabilitation for people with low vision,” *Cochrane Database of Systematic Reviews,* vol. 2, 2020.

[5] V. G and R. G, “Orientation and mobility training for adults with low vision,” *Cochrane Database of Systematic Reviews,* vol. 5, 2010.

[6] Xu, Wenyao, F. Gong, L. He and M. Sarrafzadeh, “Wearable assistive system design for fall prevention,” *Systems & Medical Device Plug-and-Play Interoperability,* pp. 1-8, 2011.

[7] Orcam, “Revolutionary Point & Click Reading,” Orcam, 2021. [Online]. Available: https://www.orcam.com/en/. [Accessed 1 May 2021].

[8] Coral, “Build beneficial and privacy preserving AI,” Google Coral, 2020. [Online]. Available: https://coral.ai/. [Accessed 29 08 2021].

[9] Nvidia, “Jetson Nano Developer Kit,” Nvidia, 2022. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed 14 July 2022].

[10] Intel, “Intel® Neural Compute Stick 2 (Intel® NCS2),” Intel, 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/hardware/neural-compute-stick.html. [Accessed 29 08 2021].

[11] A. Allan, “Benchmarking Edge Computing,” Alasdair Allan, 08 May 2019. [Online]. Available: https://aallan.medium.com/benchmarking-edge-computing-ce3f13942245. [Accessed 29 07 2021].

[12] PI3G, “Coral USB Accelerator,” Pi3G, 2021. [Online]. Available: https://pi3g.com/products/machine-learning/google-coral/coral-usb-accelerator/. [Accessed 14 July 2022].

[13] Raspberry Pi, “Raspberry Pi 4,” Raspberry Pi, [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Accessed 06 July 2022].

[14] H. Neukirchen, “Power consumption of Raspberry Pi 4 versus Intel J4105 system,” Helmut Neukirchen, 2021 June 07. [Online]. Available: https://uni.hi.is/helmut/2021/06/07/power-consumption-of-raspberry-pi-4-versus-intel-j4105-system/#:~:text=Raspberry%20Pi%204%20measurements&text=CPU%20load%20was%20generated%20using%20the%20stress%20command.,busy%2C%20it%20consumes%205.0%20W.. [Accessed 06 July 2021].

[15] M. Suryavansh, “Google Coral Edge TPU Board Vs NVIDIA Jetson Nano Dev board — Hardware Comparison,” Towards Dara Science, 20 April 2019. [Online]. Available: https://towardsdatascience.com/google-coral-edge-tpu-board-vs-nvidia-jetson-nano-dev-board-hardware-comparison-31660a8bda88. [Accessed 14 July 2022].

[16] Open Genus, “Central Processing Unit (CPU) vs Graphics Processing Unit (GPU) vs Tensor Processing Unit (TPU),” Open Genus, 2021. [Online]. Available: https://iq.opengenus.org/cpu-vs-gpu-vs-tpu/. [Accessed 28 07 2021].

[17] C. Y. N. P. a. D. P. N. Jouppi, “Motivation for and Evaluation of the First Tensor Processing Unit,” *IEEE Micro,* vol. 38, no. 3, pp. 10-19, 2018.

[18] A. M. D. E. a. A. S. D. Shadrin, “Designing Future Precision Agriculture: Detection of Seeds Germination Using Artificial Intelligence on a Low-Power Embedded System,” *IEEE Sensors Journal,,* vol. 19, no. 23, pp. 1-10, 2019.

[19] S. Sen, “Multilayer Perceptron model vs CNN,” Medium, 5 August 2020. [Online]. Available: https://medium.com/the-owl/multilayer-perceptron-model-vs-cnn-5be5cf87897a#:~:text=not%20linearly%20separable.-,Multilayer%20Perceptron%20(MLP),connected%20with%20every%20other%20perceptron.. [Accessed 2 May 2021].

[20] W. Y. S. P. F. L. Zewen Li, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE,* 2020.

[21] J. a. F. A. Redmon, “YOLO: Real-Time Object Detection,” *arXiv preprint arXiv,* vol. 1612.08242, 2016.

[22] “COCO Common Objects in Context,” COCO dataset, 2018. [Online]. Available: https://cocodataset.org/#home. [Accessed 04 08 2021].

[23] V. Feng, “An Overview of ResNet and its Variants,” Towards Data Science, 16 07 2017. [Online]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035. [Accessed 04 08 2021].

[24] https://www.kaggle.com/, “Kaggle,” Kaggle, [Online]. Available: https://www.kaggle.com/. [Accessed 29 08 2020].

[25] ImageNet, “ImageNet,” ImageNet, 11 March 2021. [Online]. Available: https://image-net.org/. [Accessed 29 08 2021].

[26] P. G.-M. a. a. H. H. A. J. G. José J. Vallés, “Three-dimensional object detection under arbitrary lighting conditions,” *Applied Optics,* vol. 45, no. 21, pp. 5237-5247, 2006.

[27] W. H. I. Y. H. G. a. a. Q. T. L. Li, “Foreground object detection from videos containing complex background,” in *Proceedings of the eleventh ACM international conference on multimedia*, 2003.

[28] I. M. a. a. I. K. P. Nousi, “Embedded UAV Real-Time Visual Object Detection and Tracking,” *IEEE International Conference on Real-time Computing and Robotics (RCAR),* 2009.

[29] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture of monkey striate cortex,” *The journel of Physiology,* vol. 195, no. 1, pp. 215-243, 1968.

[30] K. F. &. S. Miyake, “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition,” *Competition and Cooperation in Neural Nets,* vol. 45, pp. 267-285, 1982.

[31] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[32] T. M. T. ,. C. Y. ,. S. W. ,. P. S. ,. M. S. N. B. C. V. E. ,. A. A. S. A. ,. a. V. K. A. Md Zahangir Alom, “The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches,” *arXiv,* 2018.

[33] Zeiler, M. D and R. Fergus, “Visualizing and Understanding Convolutional Networks,” *arXiv,* 2013.

[34] X. Z. S. R. J. S. Kaiming He, “Deep Residual Learning for Image Recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[35] dshahid380, “Convolutional Neural Network,” Towards Data Science, 25 Feb 2019. [Online]. Available: https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529. [Accessed 29 April 2022].

[36] IBM Cloud Education, “Convolutional Neural Networks,” IBM, 20 October 2020. [Online]. Available: https://www.ibm.com/cloud/learn/convolutional-neural-networks. [Accessed 29 April 2022].

[37] A. E. Guissous, “Skin Lesion Classification Using Deep Neural Network,” 2019.

[38] IBM Cloud Education, “Neural Networks,” IBM, 17 August 2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks. [Accessed 29 April 2022].

[39] Y. LeCun, “Gradient-Based Learning Applied to Document Recognition,” in *IEEE*, 1998.

[40] MuhammadRizwan, “LeNet-5 - A Classic CNN Architecture,” Data science centre, 16 October 2018. [Online]. Available: https://www.datasciencecentral.com/lenet-5-a-classic-cnn-architecture/. [Accessed 29 April 2022].

[41] “ImageNet Classification with Deep Convolutional Neural Networks,” *Advances in Neural Information Processing Systems,* vol. 25, pp. 1097-1105, 2012.

[42] Y. Z. L. C. a. L. Z. Xiaobing Han, “settings Open AccessArticle Pre-Trained AlexNet Architecture with Pyramid Pooling and Supervision for High Spatial Resolution Remote Sensing Image Scene Classification,” *Remote Sensing ,* vol. 9, no. 8, p. 848, 2016.

[43] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv, 2014.

[44] R. Thakur, “Step by step VGG16 implementation in Keras for beginners,” Towards Data Science, 6 Aug 2019. [Online]. Available: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c. [Accessed 29 April 2022].

[45] A. G. Howard, M. Zhu, B. Chen, D. a. W. W. Kalenichenko, T. Weyand, M. Andreetto and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” in *arXiv.org perpetual*.

[46] A. Mohan, “Review On MobileNet v1,” Medium, 11 June 2020. [Online]. Available: https://medium.datadriveninvestor.com/review-on-mobilenet-v1-abec7888f438. [Accessed 23 April 2022].

[47] A. Pandey, “Depth-wise Convolution and Depth-wise Separable Convolution,” Medium, 9 Sep 2018. [Online]. Available: https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec. [Accessed 23 April 2022].

[48] K. Nganga, “Building A Multiclass Image Classifier Using MobilenetV2 and TensorFlow,” Section, 7 April 2022. [Online]. Available: https://www.section.io/engineering-education/building-a-multiclass-image-classifier-using-mobilenet-v2-and-tensorflow/. [Accessed 29 April 2022].

[49] S. Ulzhalgas, A. Daryn, I. Lyazzat and M. Eric, “Real-Time and Accurate Drone Detection in a Video with a Static Background,” *Resarch gate,* vol. 20, 2020.

[50] Bohra and Yash, “The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks,” Analytics Vidhya, 18 June 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/. [Accessed 4 April 2022].

[51] C.-F. Wang, “The Vanishing Gradient Problem,” Towards Data Science , 8 January 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. [Accessed 4 April 2022].

[52] Brownlee and Jason, “How to Fix the Vanishing Gradients Problem Using the ReLU,” Machine Learning Mastery, 11 January 2019. [Online]. Available: https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/. [Accessed 4 April 2022].

[53] S. SHARMA, “Activation Functions in Neural Networks,” Towards Data Science, 7 Sep 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed 28 April 2022].

[54] Wikipedia, “Rectifier (neural networks),” Wikipedia, 15 April 2022. [Online]. Available: https://en.wikipedia.org/wiki/Rectifier_(neural_networks). [Accessed 23 April 2022].

[55] K. Fukushima, “Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements,” *IEEE Transactions on Systems Science and Cybernetics,* vol. 5, no. 4, pp. 322-333, 1969.

[56] W. I. A. G. a. S. M. Chigozie Enyinna Nwankpa, Activation Functions: Comparison of Trends in Practice and Research for Deep Learning, arXiv, 2018.

[57] J. Feng and S. Lu, “Performance Analysis of Various Activation Functions in Artificial Neural Networks,” *Journal of Physics Conference Series,* vol. 1237, no. 2, pp. 22-33, 2019.

[58] “Incorporating Second-Order Functional Knowledge for Better Option Pricing,” *Advances in Neural Information Processing Systems,* vol. 13, pp. 472-478, 2002.

[59] J. Brownlee, “Impact of Dataset Size on Deep Learning Model Skill And Performance Estimates,” Machine Learning Mastery, 2 January 2019. [Online]. Available: https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/. [Accessed 25 April 2022].

[60] Wikipedia, “TensorFlow,” Wikipedia, 1 May 2022. [Online]. Available: https://en.wikipedia.org/wiki/TensorFlow. [Accessed 25 April 2022].

[61] PyTorch, “From Research To Production,” PyTorch, [Online]. Available: https://pytorch.org/. [Accessed 25 April 2022].

[62] Wikipedia, “Keras,” Wikipedia, 5 April 2022. [Online]. Available: https://en.wikipedia.org/wiki/Keras. [Accessed 25 April 2022].

[63] Anaconda, “Anaconda Distribution,” Anaconda, 2022. [Online]. Available: https://www.anaconda.com/products/distribution. [Accessed 25 April 2022].

[64] PyCharm, “PyCharm,” PyCharm, 2022. [Online]. Available: https://www.jetbrains.com/pycharm/promo/?source=google&medium=cpc&campaign=14124132168&term=pycharm&gclid=Cj0KCQjw1N2TBhCOARIsAGVHQc755hy8nSwlG2Ij2w_B_fMPZjYdnDsy7jxEbmvstIZPKReR52r7sgcaAsQ4EALw_wcB. [Accessed 25 April 2022].

[65] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *Arxiv: Computer Vision and Pattern Recognitio,* 2017.

[66] Pytorch Team, “MOBILENET V2,” Pytorch , [Online]. Available: https://pytorch.org/hub/pytorch_vision_mobilenet_v2/. [Accessed 29 April 2022].

[67] R. S. TIWARI, “Transfer Learning — Part — 4.0!! VGG-16 and VGG-19,” Becoming Human: Artificial Intelligence Magazine, 1 October 2021. [Online]. Available: https://becominghuman.ai/transfer-learning-part-4-0-vgg-16-and-vgg-19-d7f0045032de. [Accessed 29 April 2022].

[68] Pytorch Team, “INCEPTION_V3,” Pytorch , [Online]. Available: https://pytorch.org/hub/pytorch_vision_inception_v3/. [Accessed 29 April 2022].

[69] British Heart Foundation, “Walks and treks FAQs,” 26 June 2021. [Online]. Available: https://www.bhf.org.uk/how-you-can-help/events/training-zone/walking-training-zone/walking-faqs.

[70] Nvidia, “ADVANCED EMBEDDED SYSTEMS FOR EDGE COMPUTING,” Nvidia, 2021. [Online]. Available: https://www.nvidia.com/en-au/autonomous-machines/embedded-systems/. [Accessed 29 08 2021].

[71] E. M. C. Y. K. W. T. Qiwei Li, “Perception of Falls and Confidence in Self-Management of Falls among Older Adults,” *Int J Environ Res Public Health,* vol. December, no. 16, pp. 50-54, 2019.

[72] B. T. A, D. YA, P. N. D, L. ML, S. N. M, d. S. R. A and A. E. Jr, “Factors associated with lower gait speed among the elderly living in a developing country: a cross-sectional population-based study,” *BMC Geriatr,* vol. 15, no. 35, 2015.

[73] D. Shadrin, A. Menshchikov, D. Ermilov and A. Somov, “Designing Future Precision Agriculture: Detection of Seeds Germination Using Artificial Intelligence on a Low-Power Embedded System,” *IEEE Sensors Journal,,* vol. 19, no. 23, pp. 1-10, 2019.

[74] J. J. G., J. Vallés, P. García-Martínez and a. H. H. Arsenault, “Three-dimensional object detection under arbitrary lighting conditions,” *Applied Optics,* vol. 45, no. 21, pp. 5237-5247, 2006.

[75] L. Li, W. Huang, I. Y. H. Gu and a. Q. Tian, “Foreground object detection from videos containing complex background,” in *Proceedings of the eleventh ACM international conference on multimedia*, 2003.

[76] T. Rosen, K. A. Mack and R. K. Noonan, “Slipping and tripping: fall injuries in adults associated with rugs and carpets,” *NCBI,* vol. 5, no. 1, pp. 61-69, 2013.

[77] Zeiler, M. D and R. Fergus, “Visualizing and Understanding Convolutional Networks,” *arXiv,* 2013.

[78] Z. Li, W. Yang, S. Peng and F. Liu, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE,* 2020.